MASTER EN SCIENCES INFORMATIQUES

Utilisation de contraintes syntaxico-sémantiques dans un système de compréhension du discours continu

Mousel, Pierre

Award date:
1983

Awarding institution:
Universite de Namur

- INSTITUT D'INFORMATIQUE DES FUNDP DE NAMUR

- CENTRE DE RECHERCHES EN INFORMATIQUE
DE L'UNIVERSITE DE NANCY

UTILISATION DE CONTRAINTES SYNTAXICO-SEMANTIQUES DANS UN SYSTEME DE COMPREHENSION DU DISCOURS CONTINU

MEMOIRE PRESENTE EN VUE DE L'OBTENTION
DU GRADE DE

LICENCIE ET MAITRE EN INFORMATIQUE

PAR

MOUSEL PIERRE

A NAMUR EN

1983

A Josiane,
à mes parents

Avant-propos

Depuis leurs débuts, les recherches dans le domaine de la reconnaissance automatique de la parole n'ont cessé de progresser, surtout pendant la dernière quinzaine d'années. Les résultats qui ont été obtenus sont dus à des faits intervenant à différents niveaux. Au niveau acoustico-phonétique, la puissance des algorithmes de traitement du signal acoustique va en croissant, en particulier depuis l'introduction de la programmation dynamique, utilisée pour la normalisation dans le temps. A un niveau plus élevé, qu'on pourrait appeler syntaxico-sémantique, l'utilisation de connaissances linguistiques de haut niveau s'est révélée très utile.

C'est ce dernier aspect que nous allons étudier plus en détail tout au long de ce mémoire. Nous allons voir comment mettre en oeuvre des connaissances sur la structure du langage dans un système de reconnaissance automatique de la parole. Pour cela, certains concepts de base seront définis dans une première partie. Nous verrons ce qu'est en fait un système communication vocale homme-machine, nous verrons l'évolution des systèmes de reconnaissance automatique de la parole pendant les trente dernières années ainsi que les conditions dans lesquelles ils sont employés. Ce bref historique sera suivi du survol des principaux domaines de la linguistique dont les connaissances sont utilisées pour la reconnaissance de la parole. Quelques stratégies et organisations possibles de systèmes sont ensuite présentées avant l'exposé des moyens de mise en oeuvre de connaissances syntaxico-sémantiques dans ces systèmes.

Pour illustrer tout cela, une étude de cas sera effectuée dans une deuxième partie. Cette étude sera consacrée au système MYRTILLE II, développé au Laboratoire d'Intelligence Artificielle du Centre de Recherches en Informatique de l'Université de Nancy. C'est à la réalisation de deux procédures de ce système que l'auteur a participé dans le cadre d'un stage effectué à Nancy durant l'année académique 82/83. Ces procédures sont une procédure de création de réseau à noeuds procéduraux et une procédure d'analyse syntaxique. Elles seront présentées dans cette deuxième partie, les détails de l'analyse et de la programmation pouvant être retrouvés en annexe.

Enfin, l'auteur tient à remercier Messieurs Jacques Berleur, Jean-Paul Haton et Jean-Marie Pierrel, sans l'aide desquels ce mémoire ne serait pas.

0. TABLES

0.1. Table des matières

	Page
Avant-propos	1
0. Tables	3
0.1. Table des matières	3
0.2. Table des figures	7
PARTIE I	8
1. Introduction	9
1.1. Systèmes de communication vocale homme-machine	10
1.2. Historique des systèmes de reconnaissance automatique de la parole	13
1.2.1. Pré-histoire	14
1.2.2. Les reconnaisseurs de mots isolés	17
1.2.3. La reconnaissance du discours continu	19
1.2.4. Etat actuel des recherches	21
1.3. Les buts poursuivis, les avantages et les inconvénients	23
2. Classification des éléments constitutifs d'un langage	26
2.1. Les différentes connaissances sur les langages	27
2.1.1. La phonologie	29
2.1.2. La prosodie	31
2.1.3. La phonématique	32
2.1.4. La phonétique	33
2.1.5. La syntaxe	35
2.1.6. La sémantique	36
2.1.7. La pragmatique	37
2.1.8. Le lexique, la morphologie	38

	Page
2.2. Classification des langages selon l'importance des éléments constitutifs	40
2.2.1. Les langues naturelles	41
2.2.2. Les langages pseudo-naturels	42
2.2.3. Les langages artificiels	43
2.2.4. Informations liées à la structure du langage	44
2.2.5. Informations liées à l'application	45
2.2.6. Informations liées à la parole et/ou au locuteur	46
3. Les diverses stratégies d'utilisation des informations	47
3.1. Les méthodes d'analyse et de traitement	48
3.1.1. Les méthodes d'analyse ascendantes	49
3.1.2. Les méthodes d'analyse descendantes	50
3.1.3. Les méthodes d'analyse mixtes	51
3.1.4. Traitement gauche-droite du signal	52
3.1.5. Traitement du milieu vers les côtés	53
3.2. Organisation interne d'un système de reconnaissance de la parole	54
3.2.1. Organisation non-hiérarchisée	55
3.2.2. Organisation hiérarchisée	55
3.2.3. Organisation pseudo-parallèle	56
3.3. Les stratégies de recherche d'une solution	57
3.3.1. Stratégies totales	58
3.3.2. Stratégies heuristiques	59
4. Modèles de représentation des langages pseudo-naturels	61
4.1. Les modèles purement syntaxiques	62
4.1.1. Les grammaires hors-contexte de Chomsky	62
4.1.2. Les matrices d'adjacence	64
4.1.3. Les automates d'états finis	64
4.1.4. Les grammaires transformationnelles	65
4.2. Les modèles syntaxico-sémantiques	66
4.2.1. Les grammaires sémantiques	67
4.2.2. Les grammaires par cas	67
4.2.3. Les grammaires systémiques	68
4.3. Les modèles lexicaux et/ou sémantiques	71
4.3.1. Modèles lexicaux sur base de la définition de transitions entre les mots	72
4.3.2. Modèles lexicaux sur base de la définition de fonctions liées aux mots	72

	Page
PARTIE II	74
5. Description du système MYRTILLE II	75
5.1. Le système acoustico-phonétique	76
5.1.1. La segmentation	76
5.1.2. L'identification des segments	76
5.1.3. Le lissage de la chaîne phonémique	77
5.2. Le système syntaxico-sémantique	78
5.2.1. L'architecture de MYRTILLE II	78
5.2.2. Fonctionnement de MYRTILLE II	80
6. Description des structures syntaxico-sémantiques du langage dans MYRTILLE II	86
6.1. Objectifs et choix de réalisation	87
6.1.1. Les objectifs principaux	87
6.1.2. Les choix de réalisation	87
6.2. Définition des RNP	88
6.2.1. Les noeuds procéduraux	89
6.2.2. Les branches linéaires	89
6.3. Traitements associés aux RNP	90
7. Procédure de création d'un RNP	91
7.1. Le rôle de cette procédure	91
7.2. Les spécifications de cette procédure	91
7.3. L'implémentation de cette procédure	92
8. Procédure de parcours d'un RNP	93
8.1. Le rôle de cette procédure	93
8.2. Les spécifications de cette procédure	93
8.3. L'implémentation de cette procédure	94
9. Conclusion	95

	Page
ANNEXES	Ann. 0
ANNEXE I : Dossier d'analyse de CRERES	Ann. 1
ANNEXE II : Listing et résultats de CRERES	Ann. 44
ANNEXE III : Manuel d'utilisation de CRERES	Ann. 93
ANNEXE IV : Dossier d'analyse d'ANSYNT	Ann. 114
ANNEXE V : Listing d'ANSYNT	Ann. 136

0.2. Table des figures

	Page
fig. 1.1. Structure d'un système de communication	10
fig. 5.1. Structure générale de MYRTILLE II	79
fig. 6.1. Exemple de RNP : Le réseau GN de MYRTILLE II	88
fig. A.II.1. : Exemple de réseau à noeuds procéduraux	Ann. 89/90

PARTIE I

Cette partie sera consacrée à l'explication de toutes les notions utiles et nécessaires à la réalisation pratique, au niveau syntaxico-sémantique, d'un système de reconnaissance automatique de la parole, c.à.d. un ensemble structuré, organisé, conçu par l'esprit, coordonné de fonctions tendant à obtenir un résultat, qui est la reconnaissance de la parole, et formant à la fois une construction théorique et une méthode pratique. La parole est comprise ici comme le fait de communiquer la pensée par un système de sons articulés émis par les organes de la phonation et est opposée à l'écriture qui est l'autre moyen de mise en oeuvre d'un langage. Reconnaissance automatique signifie qu'on veut, à partir du signal vocal, émis par les organes de la phonation d'un émetteur, et d'un certain nombre de méthodes et de modèles, établir une représentation interne dans la machine, jouant le rôle de récepteur, qui permette d'interpréter ce signal et d'y répondre éventuellement par une action précise.

1. INTRODUCTION

La reconnaissance automatique de la parole n'étant qu'un axe des recherches qui se font aujourd'hui en informatique sur la parole, l'autre étant la production automatique ou synthèse de la parole, j'essaierai de délimiter clairement ces deux domaines.

Je donnerai ensuite un aperçu historique sur les réalisations qui ont été faites en reconnaissance de la parole, aperçu dont j'essaierai de dégager les buts poursuivis aujourd'hui.

1.1. Systèmes de communication vocale homme-machine

Afin de délimiter le cadre des recherches en informatique sur la parole, il importe de définir ce qu'est un système de communication vocale homme-machine. Partons de la définition la plus générale d'un système de communication (1).

La figure 1.1. décrit de manière générale un système de communication :

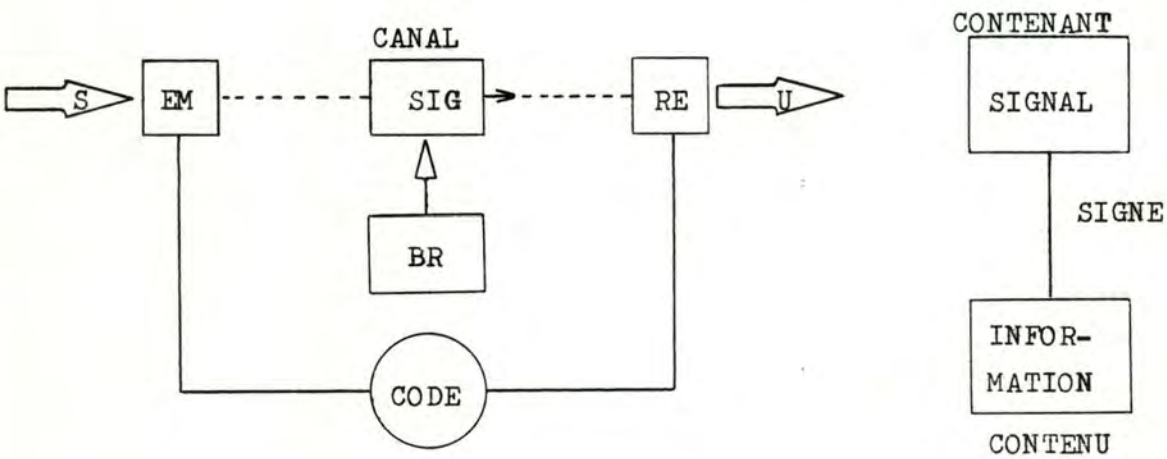


fig. 1.1. Structure d'un système de communication

---

(1) La figure ainsi que les définitions générales d'un système de communication sont extraites de l'ouvrage collectif par :

Herrlitz W. & Hundsnurscher F. en coll. avec Baumgärtner K. & Steger H.
Eine Einführung in die moderne Linguistik
Deutsches Institut für Fernstudien an der Universität Tübingen
Verlag Julius Beltz, Weinheim 1971

Par communication on comprend l'échange d'informations entre entités capables d'envoyer et/ou de recevoir de l'information. La direction de ce 'transport' d'informations détermine le récepteur et l'émetteur.

Un système de communication est donc un ensemble structuré d'entités dont le but est l'échange d'informations.

Est émetteur toute entité qui envoie de l'information, est récepteur toute entité qui reçoit de l'information (EM resp. RE).

Le CANAL est la liaison matérielle entre un émetteur et un récepteur, pouvant transmettre des signaux de l'émetteur vers le récepteur.

Le signal (SIG) est l'état matériel d'un canal et, comme tel, porteur potentiel d'information.

L'émetteur émet une information, issue d'une certaine source (S), que le récepteur destine à une certaine utilisation (U).

Les bruits (BR) sont tous les facteurs capables d'empêcher ou de gêner la transmission du signal sur le canal.

Le SIGNE est une entité, dans laquelle, à un signal, est associée une information précise. Cette association se fait par convention.

Le CODE d'un système de communication est l'inventaire des signes de ce système, en relation avec les règles de combinaison des signes de cet inventaire.

Un système de communication vocale est un système de communication où le canal est constitué d'une atmosphère pouvant transmettre des ondes acoustiques et le signal transmis est un signal acoustique représentant une chaîne de sons, déterminée par le code utilisé, ici un langage.

Un système de communication vocale homme-machine est un système de communication vocale où l'entité récepteur (ou émetteur) est un homme et l'entité émetteur (ou récepteur) est une machine.

Toutes les recherches sur l'implémentation de machines comme émetteurs dans un tel système de communication vocale homme-machine sont du domaine de la recherche sur la production automatique de la parole, encore appelée synthèse de la parole, tandis que les recherches sur l'implémentation de machines comme récepteurs dans un tel système sont du domaine de la recherche sur la reconnaissance automatique de la parole.

Précisons ces concepts :

- Reconnaissance de mots isolés : Dans un tel système, le locuteur ne parle pas d'une manière naturelle, mais doit marquer après chaque mot émis un certain temps d'arrêt, ce qui permet au système de bien délimiter les frontières de mots composant une phrase.
- Reconnaissance continue par mots clés : Dans ce cas, le système recherche dans la chaîne acoustique émise certains mots clés qui lui permettront de reconnaître l'énoncé.
- Reconnaissance continue avec contraintes sévères : Ce système permet au locuteur d'émettre des sons d'une manière naturelle (sans temps d'arrêt), mais lui impose un vocabulaire et une syntaxe réduits.
- Compréhension restreinte de phrases : Il s'agit de systèmes de reconnaissance continue travaillant sur une syntaxe proche des langues naturelles, des limitations dans le vocabulaire étant imposées par un contexte.
- Reconnaissance continue et autonome de la parole : Ce serait un système capable de reconnaître n'importe quelle chaîne de sons émise, même des suites de sons sans aucun sens. La réalisation d'un tel système est reconnue comme impossible ou, en tout cas, très difficile, selon les chercheurs.

1.2. Historique des systèmes de reconnaissance automatique de la parole

L'examen de la 'pré-histoire' des systèmes de reconnaissance automatique de la parole (qu'on appellera parfois SRP par la suite) permet de dégager les tendances principales qui se sont développées depuis le début des recherches.

Un aperçu sur l'évolution des systèmes de reconnaissance de mots isolés et de celle des systèmes de reconnaissance du discours continu nous permettra de décrire l'état actuel des recherches en mentionnant les principaux domaines d'application des SRP et servira de point de départ à l'analyse des buts poursuivis en reconnaissance de la parole.

L'histoire des SRP est celle de l'évolution d'un compromis entre besoins et désirs de l'homme et des possibilités de l'ordinateur. Cette évolution est caractérisée par une complexité croissante des systèmes dans le temps, comme le résume la figure 1.2. (1) :

évolution des SRP ↓

- Reconnaissance de mots isolés
- Reconnaissance continue par mots clés
- Reconnaissance continue avec contraintes sévères
- Compréhension restreinte de phrases
- Reconnaissance continue et autonome de la parole

fig. 1.2. Evolution des SRP

---

(1) Un très bon historique de l'évolution des SRP peut être trouvé dans l'article par :

Lea W. A.
Speech recognition: Past, Present, and Future in Trends in Speech Recognition, Lea W. A. ed., Prentice Hall, New Jersey 1980, p 39-98

1.2.1. Pré-histoire

Le problème de la reconnaissance automatique de la parole est en fait un problème interdisciplinaire. Il suppose des connaissances dans les domaines des études du langage, du son, de la physiologie, de la psychologie, des automates etc.

Certains domaines étant déjà à l'étude depuis des siècles, ce n'est cependant que vers le début du XX^{ème} siècle que les premiers résultats pratiques, utiles à la reconnaissance automatique de la parole, furent obtenus. En effet on reconnut à cette époque que le signal vocal pouvait être étudié comme tous les autres phénomènes ondulatoires. Il pouvait dès lors être décomposé en plusieurs signaux sinusoïdaux simples par une analyse du spectre des fréquences. Cette décomposition était réalisée par un certain nombre de filtres électroniques. Ce sont ces connaissances qui ont servi de point de départ à la reconnaissance automatique de la parole.

Dès le début du développement des recherches en reconnaissance automatique de la parole, deux tendances principales se sont profilées :

- Certains chercheurs trouvaient qu'il était nécessaire de comprendre comment le signal vocal était produit par les organes de la phonation de l'homme et qu'il était nécessaire d'utiliser les distinctions que l'homme utilise en produisant et en percevant différentes voyelles et consonnes (1).
- D'autres, plus orientés ingénieurs ou mathématiques suggéraient qu'on devrait plutôt faire des comparaisons ou corrélations entre les signaux à analyser et des échantillons mémorisés.

---

(1) Un exemple d'étude de ces phénomènes peut être trouvé dans l'ouvrage de :

Haton J.-P. & Pérennou G.
Reconnaissance Automatique de la parole
Actes de la 8ème école d'été d'informatique de l'AFCET
Namur 10-22 Juillet 1978

Les premiers adoptèrent donc un point de vue plus orienté linguistique que les seconds. En effet, les linguistes ont bien mis en évidence le caractère de double articulation du langage (1), une première articulation s'ordonne en unités minima sémantiques ou de sens, une seconde en unités successives minima de fonctions uniquement distinctives. Les unités de première articulation sont généralement appelés 'morphèmes' ou 'monèmes', celle de seconde 'phonèmes'. L'étude des organes de phonation de l'homme sert en fait à dégager les traits pertinents des phonèmes en vue de leur différenciation. A partir des phonèmes on peut ensuite essayer de reconstruire les morphèmes et ainsi la phrase. Les méthodes adoptant ce point de vue sont appelées 'méthodes analytiques', celles orientées plutôt mathématiques sont appelées 'méthodes globales'.

---

(1) Ce trait est, paraît-il, celui qui distingue spécifiquement le langage humain, compris comme l'ensemble des langues naturelles parlées par les hommes. En fait, il semblerait que les langues naturelles soient les seuls codes à être construits sur une codification systématique à deux étages. Un premier étage constituant une articulation suivant des unités minimales dites de signification ou significatives ('monèmes' ou 'morphèmes'), unités à deux faces : une face signifiante et une face signifiée. Grâce à ces unités significatives et un certain nombre de règles de combinaisons de ces unités, il devient possible de construire infiniment plus de messages que si chaque message devait être constitué d'un signifiant différent. Ces unités sont construites à leur tour, et on se trouve alors au deuxième niveau d'articulation, à l'aide d'unités plus petites, non-signifiantes mais uniquement distinctives, les 'phonèmes'. Ces phonèmes, toujours en nombre très réduit dans une langue, permettent de construire une infinité de monèmes, une impossibilité si chaque monème devait posséder un signifiant entièrement distinct de tous les autres. A l'aide de cette double économie dans la structuration, on arrive à construire une quantité extraordinaire de messages. Pour plus de détails sur la double articulation du langage, le lecteur pourra consulter :

Martinet A.
La double articulation linguistique dans
Travaux du Cercle Linguistique de Copenhague, 5, 1949, op. cit. in

Mounin G.
Introduction à la sémiologie
Editions de minuit, coll. Le sens commun, Paris 1970

Malgré les différences, un point de vue commun a cependant été adopté par tous les chercheurs en reconnaissance de la parole (1) :

'Un SRP doit être un ordinateur ou une machine acceptant des commandes ou des questions introduites par téléphone ou micro et générant une réponse ou action appropriée.'

Ainsi, dès le début, les chercheurs furent confrontés aux problèmes qu'on rencontre encore aujourd'hui dans ce domaine, à savoir : les problèmes posés par des locuteurs différents, le bruit et la parole continue. Très tôt on a donc commencé à limiter le problème, c.à.d. : N'admettre qu'une population réduite de locuteurs, imposer une bonne isolation acoustique, ne reconnaître que des mots isolés et un vocabulaire réduit.

Pour terminer, mentionnons encore brièvement la différence que font certains chercheurs entre reconnaissance et compréhension de la parole.

- Dans la reconnaissance de la parole, on s'efforce de reconnaître absolument tous les phonèmes dans la chaîne acoustique.
- Dans la compréhension de la parole, on s'attache plus au contenu global de la phrase en vue de générer une réponse appropriée.

La plupart des réalisations étant du domaine de la compréhension de la parole, et la reconnaissance au sens strict étant reconnue comme très difficile ou même impossible, je ne ferai plus de distinction entre les deux conceptions par la suite. Cependant comme l'indique le titre du mémoire, l'approche adoptée par l'auteur est orientée compréhension de la parole.

---

(1) art. cit. page 13 (1), p. 60

1.2.2. Les reconnaisseurs de mots isolés

Mettons brièvement en évidence les principales étapes de l'évolution de ces systèmes sans aller dans le détail des différentes réalisations (1).

Il est assez amusant de voir que le premier outil de reconnaissance était un jouet appelé 'RADIO REX'. C'était un chien qui répondait quand on l'appelait par son nom 'REX'. Il n'écoutait 'que la voix de son maître', mais répondait aussi quand son maître prononçait des mots analogues à son nom comme par exemple 'TEX'.

Ce n'est qu'en 1950 qu'ont lieu les premières tentatives sérieuses. On voit l'apparition du 'Stenosomograph', réalisé par Dreyfus-Graf. Il s'agit d'un dispositif à six filtres, transformant le signal acoustique en signal électronique projeté sur un tube cathodique.

L'année 1952 voit l'apparition du premier appareil de reconnaissance de la parole. Il a été réalisé par l'équipe Davis, Biddulph & Balashek des Bell Telephone Laboratories. Ce 'reconnaisseur' avait un taux de reconnaissance de 97%, acceptait un vocabulaire de 10 mots, était monolocuteur et orienté signal acoustique.

En 1958 Dudley et Balashek réalisent un système intéressant. Le système 'AUDREY' est en effet le premier à utiliser la segmentation des mots en phonèmes. Il donne une reconnaissance presque parfaite en mode monolocuteur.

C'est pendant les années 1959/60 que pour la première fois on utilise l'ordinateur digital pour la reconnaissance de la parole. Il est utilisé par Denes et Mathews pour faire de la normalisation dans le temps (2).

---

(1) art. cit. page 13 (1)

(2) Avant de comparer le signal acoustique d'entrée aux différents échantillons mémorisés, on peut essayer de faire correspondre la durée de ce signal à celle des différents échantillons. Ce processus s'appelle 'normalisation dans le temps' et s'est révélé très utile dans certains systèmes. Pour plus de détails, le lecteur pourra consulter :

Des contraintes syntaxiques sont utilisées par Vicens et Tubach en 1969/70.

Ce n'est qu'en 1972 qu'apparaissent les premiers produits commerciaux. Ils sont vendus par la Scope Electronics Incorporated et la Threshold Technology Incorporated.

En 1975 finalement, Itakura introduit la programmation dynamique pour la normalisation dans le temps.

Depuis, les développements vont dans le sens d'une augmentation du volume du vocabulaire et de la population des locuteurs.

- Levinson S. & Liberman M.
  La reconnaissance de la parole par ordinateur
  Pour la science, Juin 1981, p. 88-101
- Wellekens Chr. J.
  Un aperçu général sur les algorithmes de reconnaissance de la parole
  Exposé à la journée ATHENA le 17 Mai 1983 à Louvain-la-Neuve

1.2.3. La reconnaissance du discours continu

D'après Lea W. A., personne n'a jamais pu clairement déterminer si le discours continu est souhaité ou nécessaire comme moyen de communication avec l'ordinateur. Cependant fin des années soixante, début des années septante, plusieurs projets furent réalisés en vue de construire un tel système reconnaissant des mots connectés, certains travaux de base ayant déjà été faits une vingtaine d'années avant.

C'est en effet à partir des années cinquante qu'on a commencé à reconnaître la nécessité d'introduire des informations linguistiques dans le traitement de la parole. Depuis 1953 on a vu la mise en oeuvre de connaissances linguistiques, mais d'un niveau toujours assez bas.

Ce n'est qu'en 1965 que commence vraiment à se profiler une demande d'information linguistique de haut niveau.

En 1969 un point d'arrêt fut marqué par la publication du rapport Pierce (1), qui, dans son article, attaquait les chercheurs en reconnaissance de la aprole. Il ne réussit cependant qu'à retarder le début des recherches dans le domaine de la reconnaissance du discours continu et c'est en 1971 que le plus vaste projet de recherche dans ce domaine fut lancé.

---

(1) Pour plus de détails, voir :

Pierce J. R.
Whither Speech Recognition
Journal of the Acoustic Society of America, vol. 46 n= 6,
October 1969, p. 1049-1051

C'est l'Advanced Research Project Agency (ARPA) du département de la défense des Etats Unis qui fut responsable de cette reprise. Un budget de 15 millions de dollars couvrant une période de 5 ans fut dégagé pour développer des machines capables de comprendre des phrases de mots connectés faisant appel à un vocabulaire de quelques 1000 mots. Parallèlement, au début des années septante, on assiste à un grand nombre de progrès dans la technologie des ordinateurs et dans le domaine de l'intelligence artificielle, qui ont grandement facilité la mise en oeuvre pratique de ces systèmes. Le projet ARPA toucha à sa fin en 1976 avec la démonstration des systèmes HARPY, HEARSAY II, HWIM, SDC, dont certains ont même dépassé les spécifications (1).

Signalons encore qu'en même temps des recherches fructueuses furent menées aux laboratoires IBM, TI et Bell ainsi que dans certains laboratoires en Europe et au Japon (citons les systèmes MYRTILLE I et MYRTILLE II développés au laboratoire d'intelligence artificielle du centre de recherches en informatique de l'université de Nancy), aboutissant à des systèmes opérationnels avec des résultats prometteurs. Ceci nous amène à considérer l'état actuel des recherches dans ce domaine.

---

(1) Pour plus de détails sur ces systèmes, le lecteur pourra consulter les chapitres qui leur sont consacrés dans l'ouvrage :

Lea W. A. ed.
Trends in Speech Recognition
Prentice Hall, New Jersey 1980

1.2.4. Etat actuel des recherches

Ici non plus, nous ne donnons pas une liste exhaustive des systèmes actuellement sur le marché ou en étude dans les laboratoires. Bornons nous à mettre en évidence les problèmes principaux pour lesquels une solution ayant recours à des systèmes de reconnaissance de la parole est envisagée ou proposée. Notons tout d'abord que nous disposons aujourd'hui d'une technologie de reconnaisseurs de mots isolés bien établie. La marge des prix pour ces systèmes varie de 200 dollars (kit amateur) jusqu'à 80.000 dollars (systèmes professionnels). Les petits systèmes ne sont en général que des frontaux devant encore être reliés à un ordinateur, tandis que les plus grands sont généralement des systèmes de reconnaissance complets. Dans le domaine des applications commerciales, nous pouvons citer (1) :

- Systèmes de contrôle d'environnement, p. ex. : Dialog propose un tel système hospitalier pour des malades immobilisés permettant le contrôle vocal des mouvements du lit, de la climatisation, de la télé etc. Coût d'un tel système : env. 70.000 dol.
- Systèmes de tri pour les PTT ou les compagnies aériennes.
- Standards téléphoniques.
- Systèmes utilisés dans le contrôle de qualité et l'inspection, permettant à un contrôleur de vérifier la qualité d'une pièce et d'enregistrer les défauts simultanément. De tels systèmes fonctionnent dans la production de psotes de télévision, d'automobiles etc.
- Systèmes de 'Home Banking' (par téléphone), actuellement à l'étude.
- Systèmes de sécurité, permettant l'identification du locuteur par sa voix, apparemment plus fiables que d'autres systèmes.

---

(1) art. cit. page 13 (1)

La plupart de ces systèmes fonctionnent déjà dans des environnements réels et les utilisateurs rapportent généralement des économies de personnel variant de 50 à 90 %.

Enfin n'oublions pas les applications dans le domaine militaire, les militaires formant la source la plus importante de fonds pour le développement de tels systèmes (1) :

- Systèmes de cartographie
- Systèmes de contrôle du traffic aérien
- Systèmes de communication dans les cabines de pilotage
- Systèmes de surveillance de conversations
- Systèmes facilitant l'accès aux ordinateurs à des officiers supérieurs

Ayant fait un bref survol de l'évilution des systèmes de reconnaissance de la parole, connaissant les milieux d'où proviennent les principales impulsions pour ces recherches et considérant les domaines d'application les plus importants et l'effet de la mise en oeuvre de systèmes de reconnaissance de la parole dans ces domaines, essayons d'en dégager les buts poursuivis.

---

(1) art. cit. page 13 (1)

1.3. Les buts poursuivis - Les avantages et les inconvénients

Il est intéressant de remarquer que quand on se demande quels sont en fait les buts poursuivis par les chercheurs en reconnaissance automatique du discours et qu'on essaie de trouver une réponse à cette question dans des articles et ouvrages rédigés par ces chercheurs eux-mêmes, on trouvera partout des arguments humanitaires en faveur de ces recherches. Ce que l'on prétend souvent vouloir réaliser, ce sont des systèmes d'aide aux handicapés, des systèmes d'enseignement assisté par ordinateur et autres bienfaits pour l'humanité. Non que je veuille insinuer que les recherches dans ces domaines soient nulles (ne citons que les systèmes SIRENE, outil d'aide à la rééducation d'enfants malentendants et se trouvant encore dans le stade expérimental (1), et DIALOG, système de contrôle d'environnement hospitalier pour malades immobilisés), mais il est quamd même frappant de constater que la source la plus importante de fonds pour ces recherches sont les militaires dont les préoccupations sont, en général, assez éloignées de l'aide aux handicapés. Je pense donc que les buts de ces recherches ne sont pas ceux énoncés mais bien d'autres. Les systèmes militaires qu'on développe ou qu'on est en train de développer me font plutôt penser à des champs de bataille automatisés. De plus quand j'entends parler de systèmes de surveillance de conversations, cela me fait penser à '1984' de George Orwell, allez savoir pourquoi! Il est vrai cependant qu'il n'y a pas que des systèmes militaires. A côté de ceux-ci, il y a déjà un bon nombre de systèmes commercialisés. Un des multiples effets remarquables de ces systèmes est une réduction de personnel qui, chez certains utilisateurs, varie de 50 à 90 % (pour les postes de travail concernés).

---

(1) Ce système est actuellement en cours de développement au laboratoire d'intelligence artificielle du centre de recherches en informatique de l'université de Nancy. Pour une description sommaire de ce système ainsi que des autres systèmes développés dans ce centre, voir :

Rapport scientifique
Centre de recherches en Informatique de Nancy, Mars 1982

Dans une société où le travail et le produit du travail étaient équitablement répartis, cela ne poserait pas de problème. Malheureusement nous ne vivons pas dans une telle société et il suffit de regarder les statistiques de l'évolution du chômage dans le monde entier pour s'en rendre compte. Dès lors, dire que par l'implantation de tels systèmes on libère des forces qui deviennent disponibles pour d'autres tâches est absurde, parcequ'il n'y a pas d'autres tâches. De plus, la répartition équitable du produit du travail est plus que douteuse dans le monde où nous vivons actuellement. Il me semble donc que quand on travaille dans le domaine de la reconnaissance automatique de la parole, la prudence est de rigueur comme dans beaucoup d'autres domaines de recherche, car comme tout homme, le chercheur est responsable de ses actes et des effets que pourrait avoir son travail.

Indépendamment des buts pour lesquels ils sont utilisés, ces systèmes de reconnaissance de la parole présentent un certain nombre d'avantages et d'inconvénients, reconnus par la plupart des chercheurs et que l'on peut classer en trois groupes.

A) Utilisation des facultés de communication de l'homme

En effet, ces systèmes utilisent le moyen de communication qui est le plus naturel, spontané et familier à l'homme, le langage. Cependant l'homme peut émettre des occurences que la machine ne va pas reconnaître. Comme c'est le moyen le plus naturel, il ne nécessite pas d'entraînement de la part de l'utilisateur. Malheureusement ceci n'est pas le cas pour les systèmes à fortes contraintes. C'est aussi le moyen de communication le plus rapide pour l'homme mais qui est ralenti quand il s'agit de prononcer des mots isolés ou des phrases ayant une construction peu courante. Enfin ces systèmes permettent la communication multimodes entre la machine et l'homme et la communication simultanée entre un homme et un autre homme et une machine. L'inconvénient y est qu'on oublie parfois à qui on s'adresse.

B) Compatibilité avec des circonstances inhabituelles

Ces systèmes peuvent être utilisés dans le noir, à travers certains obstacles, par des aveugles et des handicapés. Ils fonctionnent même en l'absence de pesanteur, ce facteur étant cependant rarement important. De même, ils sont insensibles à de fortes accélerations ou des contraintes mécaniques. Ils permettent la vérification de l'identité du locuteur, mais sont sensibles à des dialectes ou des prononciations différentes. Grâce à eux, on peut surveiller un environnement acoustique : ce qui entraîne malheureusement qu'ils sont aussi sensibles aux bruits environnants et aux distorsions. Ils n'exigent pas de surface d'affichage ni d'appareillage compliqué, par contre un micro doit être porté en permanence.

C) Mobilité et liberté d'action

On peut utiliser ces systèmes à distance ou dans différentes positions, de même qu'on peut simultanément utiliser les yeux et les mains pour d'autres tâches. Enfin ils permettent l'utilisation de téléphones comme terminaux d'ordinateurs, à condition qu'on résolve le problème des vastes population de locuteurs, de bruits et de distorsions.

Ayant maintenant clairement défini ce qu'est un système de reconnaissance de la parole, ayant donné un bref aperçu sur l'évolution de ces systèmes depuis les années cinquante et ayant discuté les avantages et inconvénients que présente l'utilisation de tels systèmes, passons à l'examen des connaissances nécessaires pour les réaliser.

2. CLASSIFICATION DES ELEMENTS CONSTITUTIFS D'UN LANGAGE

Comme la reconnaissance de la parole est un problème interdisciplinaire, on y fait appel à un certain nombre de connaissances traitant différents aspects du langage. Après avoir proposé une classification de ces éléments constitutifs d'un langage, on essaiera de classer les langages en fonction de l'importance de chacun de ces éléments et de déterminer à quoi ces éléments sont liés.

2.1. Les différentes connaissances sur les langages

Pour réaliser un système de reconnaissance de la parole performant, c.à.d. reconnaissant le discours continu, acceptant un vocabulaire et une population de locuteurs importants, il est indispensable, et cela on l'a remarqué dès la fin des années soixante, de mettre en oeuvre un maximum de connaissances et spécialement des connaissances linguistiques de haut niveau (1). Ces connaissances deviennent alors pour un système de reconnaissance de la parole des sources d'informations auxquelles le système peut faire appel pour résoudre son problème, c.à.d. reconnaître une phrase.

Jean-Marie Pierrel propose une classification de ces sources d'informations (2), nous permettant un essai de définition de ces différents éléments du point de vue linguistique et une définition que nous appliquerons dans le cadre de la recherche sur la reconnaissance de la parole. Il distingue entre :

- Le lexique et la morphologie
- La syntaxe
- La sémantique
- La pragmatique
- La prosodie
- La phonétique
- La phonologie (- La phonématique)

Essayons maintenant de définir ces domaines du point de vue linguistique d'abord, du point de vue SRP ensuite.

---

(1) En fait J. Weizenbaum mettait déjà en évidence l'importance d'informations contextuelles dans la communication à l'aide des langues naturelles entre l'homme et la machine, et ce dans deux articles parus en 1966 et 67 :

Weizenbaum J.
ELIZA - A computer program for the study of natural language communication between man and machine
CACM vol 9 n° 1, January 1966

Weizenbaum J.
Contextual understanding by computers
CACM vol 10 n° 8, August 1967

(2) Cette classification est proposé dans l'ouvrage :

Pierrel J.-M.
Etude et mise en oeuvre de contraintes linguistiques en compréhension automatique du discours continu
Thèse présentée pour l'obtention du grade de Dr. ès-Sc. Math.(Inf.)
Nancy, Mars 1981

2.1.1. La phonologie

Pour R. Vion, comme il l'écrit dans l'Encyclopaedia Universalis (1), la phonologie se présente comme la discipline scientifique ayant pour domaine l'étude de l'aspect phonique des langues naturelles. Ce domaine en apparence bien délimité a néanmoins permis l'existance de plusieurs disciplines distinctes :

- La phonétique historique
- La phonétique générale
- La phonologie

La phonologie se distingue de l'analyse physiologique et acoustique des sons du langage qu'est la phonétique. Elle se distingue également de l'étude histirique des sons telle que la formulèrent les linguistes du XIX^{ème} siècle. C'est précisément l'existence de la phonétique, traitant déjà de l'aspect phonique des langues, qui allait obliger la phonologie à se définir en opposition à celle-ci. On pourrait dire brièvement que la phonétique étudie les sons de la parole à la manière des sciences physiques : Elle opère une analyse physique des sons du langage. Or un classement physique des sons du langage ne peut satisfaire les exigences d'une étude linguistique, car la phonétique se trouve dans l'impossibilité de décider de la fonction linguistique des différences phoniques qu'elle constate. Il y a donc place pour une discipline spécifique abordant linguistiquement l'aspect phonique des langues naturelles : la phonologie. Celle-ci doit être en premier lieu capable de décider de la valeur linguistique des différences phoniques. Dans une langue donnée, la valeur d'une différence phonique est déterminée par la manière dont elle contribue à l'établissement de la fonction communicative, c.à.d. par la manière dont elle participe à l'établissement des messages linguistiques.

---

(1) Vion R.
Phonologie
Encyclopaedia Universalis vol 12 pages 992-994

L'objet de cette discipline peut dès lors être défini comme suit : La phonologie permet d'affecter à chaque différence phonique une fonction linguistique déterminée dans la langue où elle apparaît. Elle présente ainsi une hiérarchie parmi les différences phoniques correspondant à la hiérarchie de fonctions envisagées. Cette hiérarchie possède en fait deux niveaux : d'une part, les différences phoniques assurant la fonction distinctive, d'autre part celles, qui, ne l'assumant pas, remplissent au moins l'une des autres fonctions secondaires (expressive ou démarcative). Comme il est délicat de hiérarchiser ces fonctions secondaires, la phonologie classe en fait les différences phoniques selon un seul critère : la fonction distinctive. L'objet de la phonologie est donc l'analyse de l'aspect phonique des langues du point de vue de la fonction distinctive.

Pour nous, dans le domaine de la recherche sur la reconnaissance de la parole, cette définition sera quelque peu apauvrie. Pour J.-M. Pierrel, la phonologie se ramène à l'étude des altérations possibles d'un phonème ou d'un mot suivant son contexte, à l'exclusion des aspects combinatoires de traits ou de phonèmes. Il distingue trois types d'altérations phonologiques (1) :

- Altérations phonologiques à l'intérieur d'un mot, où on peut distinguer :
  = les altérations dues à l'influence mutuelle de mouvements articulatoires voisins : assimilation, fusion, coarticulation, anticipation...
  = les altérations dues aux accents
- Altérations en fin de mots dues aux conjugaisons ou déclinaisons
- Altérations à la jonction des mots des types :
  = élisions
  = insertions
  = substitutions
  = liaisons

En traitement automatique de la parole, seul cet aspect altérations est important.

---

(1) op. cit. page 28 (2)

2.1.2. La prosodie (1)

Une des deux branches de la phonologie, toujours d'après l'Encyclopaedia Universalis, est la prosodie, l'autre étant constituée par la phonématique. La prosodie étudie les phénomènes 'suprasegmentaux', c.à.d. ceux qui ne sont pas segmentables dans le cadre de la double articulation (caractéristique du langage humain, rappelons le, et se faisant en morphèmes d'abord, en phonèmes ensuite. Cfr. page 15). On peut aussi distinguer comme faits relevant de l'approche prosodique :

- L'accent tonique, qui, par la mise en relief d'une syllabe par rapport aux autres, permet de distinguer le sens de deux suites phoniques phonétiquement semblables.
- L'intonation qui permet par exemple en français de distinguer l'affirmation de l'interrogation dans deux suites semblables du point de vue des unités de première et de seconde articulation.
- Le degré d'allongement dont la pertinence est variable. Dans la langue arabe, la longueur est un trait pertinent au sens phonologique du terme, mais dans une langue comme le français, l'allongement peut avoir une fonction expressive.

L'importance de ce qu'on englobe sous le nom de prosodie, ou encore faits suprasegmentaux, a été démontrée par des recherches en laboratoire sur la synthèse de la parole. Cette définition étant celle des linguistes, les phénomènes prosodiques qui nous intéressent dans le domaine du traitement automatique de la parole sont de deux types seulement :

- Les marqueurs prosodiques délimitant des unités successives. Ces marqueurs sont les seuls à pouvoir lever certaines ambiguïtés. Malheureusement les études de ces phénomènes ne sont pas encore assez avancées pour qu'on puisse automatiser leur traitement.
- La mélodie générale de la phrase. Ce phénomène permet de distinguer trois suites phoniques phonétiquement semblables dont une serait énonciative, l'autre interrogative et la troisième impérative. Cependant la prudence est de rigueur, car le contenu mélodique est souvent assez complexe.

(1) Prosodie
Encyclopaedia Universalis vol. 20 p 1577

2.1.3. La phonématique

La phonématique, pour les linguistes (1), est à côté de la prosodie la deuxième branche de la phonologie. Il s'agit de la partie de la phonologie qui aborde l'étude du matériel phonique ne relevant que du niveau de la deuxième articulation, elle traite donc du dégagement des phonèmes, de leur définition et de leur classement de manière à constituer le système phonologique. Elle traite également des combinaisons de phonèmes à l'intérieur des unités immédiatement supérieures : les monèmes. Mais elle n'aborde généralement pas l'étude de l'intonation ou de l'accentuation qui ne relèvent pas du même niveau d'analyse que les phonèmes. Ces faits prosodiques et accentuels sont d'ailleurs assez souvent nommés éléments marginaux ou suprasegmentaux. L'étude du point de vue de la fonction distinctive de ces éléments marginaux relève de la partie non phonématique de la phonologie, souvent appelée prosodie. Cette même partie traite également de l'étude des fonctions secondaires. Elle est donc loin de posséder l'unité de la phonématique. Elle est également loin d'avoir subi les mêmes développements qu'elle. Il en résulte que d'une manière générale, lorsqu'on parle de phonologie ou de méthode phonologique c'est en fait de phonématique qu'il s'agit. On emploie donc souvent indifféremment, mais à tort, les termes phonologie et phonématique pour parler de cette dernière. C'est en fait ce qu'a fait J.-M. Pierrel dans sa classification qui distingue entre phonologie et prosodie, ne parlant pas de la phonématique. Ce qu'il définit comme étant la phonologie est en réalité la phonématique, la phonologie regroupant les deux domaines que sont la prosodie et la phonématique.

---

(1) art. cit. page 29 (1)

2.1.4. La phonétique

La phonétique est définie (1), à peu près unanimement, comme 'l'étude des sons du langage'. Cet accord sur une formule particulièrement vague ne suffit pas à masquer les dissensions profondes qui se manifestent lorsqu'il s'agit de préciser l'objet, les méthodes, en somme le statut scientifique de cette discipline. Toute la problématique qui surgit ainsi, constitue un raccourci particulièrement représentatif des problèmes épistémologiques posés dans le contexte scientifique contemporain, par la délimitation du domaine de certaines sciences d'autonomie récente (surtout des sciences humaines) dont l'évolution des conceptions internes a été particulièrement rapide depuis le début du XX^{ème} siècle. La phonétique fait-elle partie des sciences humaines ou des sciences naturelles? Est-elle particulièrement une science abstraite - de la forme linguistique - ou exclusivement une science concrète - de la substance sonore - ? L'existence d'un aspect théorique (surtout linguistique) et d'un aspect expérimental (utilisant des moyens techniques perfectionnés et des méthodes d'investigations empruntées aux sciences physiques) remet-elle en question l'unité de cette discipline et son intégration dans le domaine linguistique? Telles sont les questions fondamentales qui se posent dans le domaine de la phonétique.

---

(1) Autesserre D.
Phonétique
Encyclopaedia Universalis vol 12 p 988-992

Pour les chercheurs dans la reconnaissance de la parole, le terme 'phonétique' regroupe un ensemble d'informations acoustico-phonétiques, J.-M. Pierrel parle même d'informations acoustico-phonémiques se décomposant en deux types d'informations :

- Les résultats du niveau de traitement acoustico phonétique (les deux grands niveaux de traitement étant le niveau acoustico-phonétique et le niveau syntaxico-sémantique (1)) dans le système de reconnaissance de la parole. Ces résultats peuvent être constitués d'une pseudo-chaîne d'éléments minimaux de type phonème ou de paramêtres divers. On parlera de pseudo-phonèmes parce que premièrement un phonéticien n'appellerait pas ces éléments minimaux des phonèmes, un phonème étant une unité abstraite n'ayant d'existence que phonologique, et deuxièmement à cause des erreurs potentielles de détermination de ces segments minimaux.
- L'ensemble des représentations phonétiques des mots du lexique.

---

(1) Les systèmes de reconnaissance du discours continu travaillent souvent en deux étapes. Dans un premier temps, le signal acoustique est soumis à un traitement de base, encore appelé traitement acoustico-phonétique. Le but de ce traitement est d'obtenir un treilli ou chaîne de pseudo-phonèmes, représentant la phrase prononcée. Les éléments de cette chaîne sont appelés pseudo-phonèmes parce que, ressemblant à des éléments de type phonème, ils ne vérifient cependant pas la définition qu'en donnerait un linguiste. De plus, des erreurs s'infiltrent lors de la prononciation de la phrase et lors de son traitement de base, ce qui est une raison supplémentaire de ne pas parler de phonèmes. En plus de cette chaîne de pseudo-phonèmes ce traitement fournit encore des paramêtres divers. Après avoir été soumis à ce traitement acoustico phonétique, la chaîne de pseudo-phonèmes est soumise à un traitement syntaxico-sémantique dont le but est la reconstruction de la phrase à l'aide de la pseudo-chaîne de phonèmes et des informations sur la structure du langage dont le système dispose.

2.1.5. La syntaxe

L'Encyclopaedie Universalis (1) nous apprend qu'en général l'étude d'une langue se fait sous trois aspects : un aspect sémantique (lexical), un aspect morphologique et un aspect syntaxique. On peut définir la syntaxe comme étant essentiellement l'étude de la combinaison des mots dans la phrase, de la construction des propositions et des rapports qu'elles entretiennent. La syntaxe se consacre à la première articulation du langage, fait la liste des monèmes et les classe selon les fonctions qu'ils peuvent remplir dans la phrase. D'où le nom de 'syntaxe fonctionnelle'. Cette syntaxe est complétée par une étude phonologique s'attachant à la deuxième articulation.

Pour des gens comme Chomsky, la syntaxe est la partie générative d'une grammaire, c.à.d. de la description complète d'une langue, en ce qu'elle engendre, selon des mécanismes purement formels, toutes les suites de morphèmes considérées comme grammaticalement correctes et uniquement ces suites. Elle peut être conçue comme un mécanisme fini engendrant un nombre infini de phrases grammaticales grâce aux processus récursifs qu'elle comprend. Les rapports entre syntaxe et sémantique sont actuellement très mal connus et compris. C'est cet aspect de mécanisme formel génératif qui nous intéresse dans le domaine de la reconnaissance de la parole. Cependant il faut encore remarquer que le niveau de détail de la spécification des règles de production de ce système formel joue un rôle très important et que le choix d'un bon niveau de description est un facteur déterminant lors de la mise en oeuvre d'un système de raconnaissance de la parole.

---

(1) Syntaxe
Encyclopaedia Universalis vol 20 p 1862

2.1.6. La sémantique

Il s'agit là d'un domaine particulièrement difficile à définir et à délimiter, selon Georges Mounin dans son article dans l'Encyclopaedia Universalis (1). Grossièrement, c'est l'étude du sens des mots. C'est l'étude des relations entre le signifiant du signe, le signifié du signe et le référent du signe.

Le linguiste Uriel Weinreich (2) la définit comme suit : 'La sémantique étudie la transmission du sens au moyen des mécanismes grammaticaux et lexicaux d'une langue.'

Après 1930, l'attention des chercheurs en linguistique s'est concentrée surtout sur l'étude des structures formelles du langage afin de rendre plus rigoureuse et plus fine la description de celui-ci. Malheureusement, les structures formelles ne sont qu'un moyen pour la communication linguistique, tandis que la fin est la transmission d'une signification, et l'étude de ces significations n'a guère avancée, comparée à celle des structures formelles. Les difficultés qu'on rencontre dans l'étude de ces significations ont bien été mises en évidence dans l'article de Chomsky : 'La forme et le sens dans le langage naturel' (3).

Pour les chercheurs dans le traitement automatique de la parole, la sémantique est essentiellement un ensemble de contraintes limitant les constructions syntaxiques possibles. Ces contraintes sont fortement liées à l'application qu'on a l'intention de réaliser, on parlera donc de la sémantique d'une application.

---

(1) Mounin G.
Sémantique
Encyclopaedia Universalis vol 14 p 854

(2) Weinreich U.
Explorations in semantic theory
Current trends in linguistics vol III, La Haye 1966
art. cit. in (1)

(3) Chomsky N.
La forme et le sens dans le langage naturel
Hypothèses, Ed Seghers, Coll. Change, Série Hypothèses, 1972

2.1.7. La pragmatique

Ce mot vient du grec 'pragma' qui signifie 'action'. On peut la définir comme un ensemble de connaissances qu'on a sur un certain sujet (1). (Le pragmatisme est la philosophie de la science, c'est un expérimentalisme. L'esprit du pragmatisme est l'esprit de laboratoire.)

Pour J.-M. Pierrel, c'est l'ensemble des connaissances à priori, que l'on possède sur l'application mise en oeuvre et recouvre aussi certains présupposés nécessaires au dialogue.

---

(1) Deledalle G.
Pragmatisme
Encyclopaedia Universalis vol 13 p 441-443

2.1.8. Le lexique - la morphologie

Ces deux concepts ont été groupés pour des raisons de mise en oeuvre pratique.

La lexicologie (1) est l'étude des mots. Elle constitue donc une des trois parties de la linguistique avec la phonologie, ou étude des sons constitutifs des mots, et la grammaire, ou étude des relations entre les mots (syntaxe) et des marques qui signifient ces relations (morphologie).

Au sens étroit (2), la morphologie est la partie de la grammaire qui s'occupe de la formation des mots par adjonction d'affixes à des thèmes. En ce sens, morphologie s'oppose essentiellement à syntaxe, cette dernière étant l'étude des rapports entre les éléments de la pjrase.

Dans le domaine du traitement de la parole sur ordinateur, le lexique va contenir l'ensemble des informations relatives aux mots. Le contenu de ce lexique peut aller du 'lexique dictionnaire', simple énumération des mots acceptés par le langage, au lexique 'ensemble homogène par niveaux contenant des informations acoustiques, phonologiques, syntaxiques et sémantiques'.

---

(1) Guiraud P.
Lexicologie
Encyclopaedia Universalis vol 9 p 943

(2) Morphologie
Encyclopaedia Universalis vol 19 p 1312

Après ce survol des aspects principaux que nous fournissent les connaissances actuelles sur les langages, tant du point de vue du linguiste que du point de vue du chercheur dans le domaine du traitement automatique de la parole, nous pouvons classifier les langages en fonction de l'importance des différents éléments et déterminer ce à quoi ces éléments sont liés.

2.2. Classification des langages selon l'importance des éléments constitutifs

L'histoire du développement des systèmes de traitement automatique de la parole nous montre que toutes les sources d'informations linguistiques telles que nous les avons exposées dans la section précédente n'ont pas été exploitées dès le début. Certains aspects sont encore assez obscurs, même aujourd'hui. La conséquence en est que les langages qui sont traités pae des systèmes de reconnaissance automatique de la parole sont encore assez éloignés des langues naturelles. Cependant l'utilisation plus ou moins poussée de connaissances linguistiques de haut niveau permet déjà de proposer une certaine classification des langages traités automatiquement.

2.2.1. Les langues naturelles

Par langues naturelles on comprend toutes les langues que les hommes apprennent à partir de leur enfance en vue de communiquer entre eux. Ces langues sont trop complexes et leurs structures sont encore trop mal maîtrisées pour qu'elles puissent être traitées automatiquement dans l'ensemble.

2.2.2. Les langages pseudo-naturels

Ce sont des langages qui ont une syntaxe souvent limitée qui se rapproche très fort de celle d'une langue naturelle, mais un lexique spécifique à un domaine précis d'application. Les avantages généralement reconnus d'un tel langage sont les suivants :

- La communication est quasi naturelle.
- L'apprentissage n'est pas nécessaire pour autant qu'on n'utilise pas de concepts qui sortent du cadre de l'application prévue. On voit ici l'importance d'un lexique bien construit.
- Le passage est facile d'une application à une autre.

Dans ce type de langage, les informations sémantiques et lexicales seront donc particulièrement importantes, car ce seront elles qui reflèteront la spécifité de l'application. Ce sont en effet ces informations qui contraignent le système, il faut donc porter un soin particulier à leur élaboration si on veut obtenir un traitement efficace et performant.

2.2.3. Les langages artificiels

Leur caractéristique est une syntaxe et un vocabulaire très rigides. Ils ne ressemblent qu'à première vue à une langue naturelle. Du fait de leur rigidité, ils demandent une phase d'apprentissage pour pouvoir être utilisés par l'homme. La définition des structures de ces langages dépendant fortement de l'application pour laquelle ils sont conçus, un changement d'application demande en général une redéfinition complète de ceux-ci.

Après cette classification des langages, selon l'importance de certains éléments constitutifs de ces langages, voyons maintenant à quoi ces éléments sont liés. Pour cela j'adopte également la classification proposée par J.M. Pierrel (1) :

- Informations liées à la structure du langage.
- Informations liées à l'application.
- Informations propres à la parole et/ou au locuteur.

Afin de réaliser un système de reconnaissance automatique de la parole efficace et performant, qui de plus est facilement modifiable, il faut essayer de prendre en compte un maximum d'informations et de voir quels sont les facteurs influençant ces informations. De cette manière on sait quelles sont les informations à changer si l'on change de langage, d'application ou de locuteur.

---

(1) op. cit. page 28 (2)

2.2.4. Informations liées à la structure du langage

La première chose qui nous vient à l'esprit lorsque nous disons 'structure du langage', c'est : 'syntaxe'. Mais, déjà, une première remarque s'impose :

- Si la définition de la syntaxe du langage ne doit pas aller plus bas dans le niveau de détail que jusqu'aux différentes classes grammaticales, alors ce n'est pas seulement dans le cadre des informations liées à la structure du langage que la syntaxe doit intervenir. En effet, à un certain moment, il sera nécessaire d'établir un lien entre les classes grammaticales et les différents mots qui constituent le lexique.
- De même l'ensemble des informations liées à la structure du langage ne se limite pas à la seule syntaxe. On y retrouve de fait :
  = Des informations lexicales regroupant essentiellement les mots grammaticaux, qui tout en étant partie intégrante du lexique, n'en restent pas moins fortement liés à la structure du langage.
  = Des informations prosodiques. La mélodie générale d'une phrase renseigne en effet souvent sur sa structure.
  = Des informations sémantiques. En effet la syntaxe du langage propose un certain nombre de choix et d'alternatives. La prise en compte d'informations sémantiques permet alors de restreindre fortement ces alternatives et le choix des structures possibles.

2.2.5. Informations liées à l'application

On peut y retrouver :

- Le lexique.
  On peut ici se reporter à la définition des unités linguistiques ou mots du langage. Cette définition comprend des aspects syntaxiques et sémantiques.
- La sémantique.
  Il s'agit non seulement de la signification de chaque mot, mais également de la définition des liaisons possibles entre les différents mots, compte tenu de l'application particulière choisie.
- La syntaxe.
  Elle intervient sous deux aspects :
  = Informations syntaxiques liées aux mots du lexique. Elles permettent de faire la liaison entre la définition de la structure du langage et les différents mots du lexique.
  = Restrictions syntaxiques portant sur la structure générale des phrases. La définition de la structure du langage étant indépendante de l'application, il faut avoir la possibilité d'éliminer certaines constructions que l'application donnée ignore.
- La pragmatique.
  Elle regroupe ici :
  = Un ensemble de connaissances a priori sur l'application.
  = Une liste de scénarios possibles de dialogues.
  = Un ensemble de valeurs par défaut pour certaines fonctions sémantiques.

2.2.6. Informations liées à la parole et/ou au locuteur

On y retrouve :

- Les informations liées au lexique.
  C'est la définition acoustico-phonétique des mots. Elle comprend :
  = La représentation acoustico-phonétique des mots.
  = Les numéros de règles d'altérations phonologiques exprimant les diverses désinences possibles du mot (dans le lexique on ne conserve que le radical de chaque mot).
  = D'autres informations plus pragmatiques, tels que la longueur phonétique et le patron phonétique du mot.
- Les informations liées au locuteur.
  Ce sont des informations rendant compte des habitudes du locuteur. Dans un système mono-locuteur ou fonctionnant faussement en multi-locuteur, c'est à ce niveau que se fera la prise en compte du lexique propre à chaque locuteur.

Ayant proposé une classification des informations pouvant être utilisées par un système de traitement automatique de la parole, ayant dégagé les éléments auxquels ces informations sont liées, passons à la description des différentes stratégies d'utilisation de ces informations en vue de la reconnaissance d'une phrase.

3. LES DIVERSES STRATEGIES D'UTILISATION DES INFORMATIONS (1)

Comme le but d'un système de reconnaissance de la parole est de reconnaître une phrase du langage accepté par ce système, en faisant appel à une multitude de connaissances sur ce langage, il est possible d'envisager l'exploitation de ces connaissances sous différents aspects et d'adopter des stratégies d'utilisation diverses de ces informations.

Il y aura lieu d'abord les méthodes d'analyse d'une phrase, puis les différentes organisations possibles des modules à l'intérieur du système et enfin les différentes manières d'avancer dans l'espace des solutions. Mais, avant tout, une remarque préliminaire s'impose :
Comme nous travaillons au niveau syntaxico-sémantique, chaque fois que nous parlerons du signal vocal, nous nous référons, en fait, à sa représentation sous forme de pseudo-phonèmes (Le signal physique ayant déjà été traité par le niveau acoustico-phonétique. Cfr. page 34 (1).).

---

(1) Source : op. cit. page 28 (2)

3.1. Les méthodes d'analyse et de traitement

Essentiellement deux approches de base se profilent dans les systèmes de reconnaissance de la parole, deux approches qu'on retrouve aussi en compilation :

- L'approche descendante, qui consiste à partir des connaissances linguistiques.
- L'approche ascendante, qui consiste à partir du signal vocal.

Ces deux méthodes d'analyse peuvent être combinées à deux méthodes de traitement du signal vocal généralement utilisées dans les systèmes de reconnaissance de la parole :

- Un traitement gauche-droite du signal
- Un traitement du milieu vers les côtés du signal

Chacune de ces méthodes présente ses avantages et ses désavantages que nous allons analyser plus en détail.

3.1.1. Les méthodes d'analyse ascendantes

Elles consistent à partir du signal vocal (c.à.d. de sa transcription sous forme de pseudo-phonèmes) et de reconnaître la phrase en remontant les différents niveaux de connaissances linguistiques, depuis les connaissances phonétiques jusqu'aux connaissances sémantiques.

Les avantages et inconvénients généralement reconnus de ces méthodes sont les suivants :

Comme pour ces méthodes, il faut construire le treillis (ou la chaîne) de tous les mots reconnaissables à partir de la pseudo-chaîne de phonèmes, ce treillis risque de devenir vite assez volumineux et la méthode très coûteuse. Par contre, le fait de disposer de l'ensemble des mots reconnaissables nous procure une relative immunité contre les erreurs de transcription de la phrase.

3.1.2. Les méthodes d'analyse descendantes

Contrairement aux méthodes d'analyse ascendantes on part ici des connaissances linguistiques de plus haut niveau, c.à.d. des connaissances sur la structure du langage (syntaxico-sémantiques). Ceci permet de prédire l'existence éventuelle d'un certain nombre de mots, existence qui sera vérifiée grâce au lexique où on trouvera la représentation phonémique des mots, laquelle pourra être comparée avec le signal vocal.

Les avantages et les inconvénients généralement reconnus de ces méthodes sont les suivants :

Grâce aux connaissances de très haut niveau desquelles on part, on élimine tout de suite un certain nombre de mots candidats dans le cas de l'analyse ascendante. Ces méthodes sont donc en général beaucoup moins coûteuses de ce point de vue. Cependant, dans les systèmes à vocabulaire volumineux, le nombre d'hypothèses à vérifier reste très élevé et on perd son temps à éliminer des constructions erronnées. Ce type d'analyse est également plus sensible aux phénomènes de bruitage. Comme on part de connaissances linguistiques de haut niveau et comme il faudra à un certain moment vérifier la pertinence des hypothèses émises grâce à ces connaissances sur le signal vocal, ces méthodes d'analyse exigent des procédures de synchronisation sur ce signal vocal.

3.1.3. Les méthodes d'analyse mixtes

Plusieurs systèmes, alin de pallier les désavantages des méthodes d'analyse ascendantes ou descendantes pures, proposent des solutions s'inspirant à la fois des unes et des autres. Ces méthodes qu'on pourrait appeler méthodes mixtes fonctionnent suivant un processus d'hypothèses-tests se basant principalement sur les méthodes descendantes.

Les apports principaux des différentes approches sont respectivement :

- Pour l'approche descendante : Elle permet d'éliminer les indéterminations liées à la structure du langage.
- Pour l'approche ascendante : Elle permet de synchroniser les traitements sur le signal, c.à.d. de déterminer les points de départ et les points de reprise du traitement sur le signal vocal.

Signalons enfin que c'est une telle méthode d'analyse mixte qui a été adoptée dans le système MYRTILLE II que nous examinerons dans notre deuxième partie.

3.1.4. Traitement gauche-droite du signal

Cette méthode semble la plus naturelle pour traiter la parole. Elle consiste en effet à traiter le signal vocal dans l'ordre de sa production du début à la fin. On utilise donc pleinement les possibilités de prédiction des connaissances sur la structure du langage.

Deux inconvénients se profilent cependant :

- Le commencement et la fin d'une phrase sont parfois difficiles à détecter.
- Les bruits perturbent fortement les traitements et provoquent de fréquents retours en arrière.

Signalons toute fois que l'analyseur du système MYRTILLE II effectue un tel traitement de la gauche vers la droite.

3.1.5. Traitement du milieu vers les côtés

C'est une méthode qui travaille à partir d'un certain nombre d'îlots de confiance (par îlot de confiance on comprend une partie de phrase qu'on est sûr d'avoir bien transcrite en pseudo-phonèmes) pour reconnaître la phrase, en essayant de la reconstruire à partir d'un certain nombre de représentations partielles. Les désavantages du traitement gauche-droite sont donc éliminés. Cependant d'autres difficultés apparaissent :

- Des problèmes de concordance entre les différentes représentations partielles.
- Des problèmes de choix des critères de sélection des îlots de confiance.

Ainsi les différentes méthodes d'analyse du signal vocal qu'un système de reconnaissance de la parole peut utiliser, nous conduisent à examiner les différentes organisations internes possibles de ces systèmes et à voir les interactions entre leurs différents modules.

3.2. Organisation interne d'un système de reconnaissance de la parole

Comme un 'reconnaisseur' de la parole est censé faire appel à une multitude de connaissances linguistiques, il faut organiser l'interaction entre les différents modules chargés de la prise en compte des différentes connaissances. En général on dispose de trois modules différents et ce pour la prise en compte des informations :

- sur la structure du langage
- lexicales
- prosodiques et phonétiques.

C'est le jeu de la mise en oeuvre de ces informations en vue de la reconnaissance d'une phrase que l'organisation doit déterminer. Tentons de donner une classification des différentes organisations qui ont été adoptées dans les systèmes de reconnaissance de la parole.

3.2.1. Organisation non hiérarchisée

Cette organisation doit permettre la mise en oeuvre des informations au fur et à mesure des besoins du traitement sans schéma préétabli. On doit donc pouvoir activer à n'importe quelle phase du traitement les modules exploitant les informations liées à la structure du langage, celles liées à l'application ou celles liées au locuteur (cfr. pages 44-46). Cependant dans les systèmes actuels ayant une organisation non-hiérarchisée (p. ex. le système HEARSAY II), on constate une structure implicite prévue par le concepteur en fonction des différents cas pouvant se présenter, l'ordinateur ne pouvant pas encore prendre certaines décisions, vu l'état actuel de nos connaissances.

3.2.2. Organisation hiérarchisée

Contrairement à l'organisation précédente, celle-ci impose une forte hiérarchie entre les modules de prise en compte des informations. Cette hiérarchie peut varier de système à système (p. ex. : MYRTILLE I utilise une hiérarchie structure-lexique alors que le système KEAL utilise une hiérarchie lexique-structure (1)). A cause de cette structure préétablie et trop rigide, les systèmes organisés de cette manière ne sont pas adaptés au traitement des langages pseudo-naturels, pour lesquels le processus de mise en oeuvre des informations varie souvent en fonction du contexte.

---

(1) Mercier G., Quinton P. & Vires R.
KEAL : un système pour un dialogue oral avec une machine
Congrès AFCET, TTI, Paris, Novembre 1978, p. 304-314
art. cit. in op. cit. page 28 (2)

Pierrel J.-M.
Un système de compréhension du discours continu utilisant des contraintes morpholigiques, syntaxiques et sémantiques
RAIRO, Informatique/Computer science, vol 12, n° 2, 1978

3.2.3. Organisation pseudo-parallèle

Afin de pallier les inconvénients des deux autres organisations, on peut choisir cette organisation où la hiérarchie est établie dynamiquement lors de l'exécution par un module superviseur. Ce superviseur décide de cette hiérarchie en fonction d'un certain nombre de critères eux mêmes hiérarchisés de manière à ce que le système dispose d'une hiérarchie par défaut. Les conséquences d'une telle organisation sont :

- L'indépendance entre les différentes définitions des sources d'informations.
- La nécessité de la compatibilité entre les résultats de la mise en oeuvre des différentes informations.

Au vu de ces différentes manières d'organiser l'enchaînement des traitements sur les différentes sources d'informations, nous pouvons examiner les façons de se déplacer dans l'espace des solutions.

3.3. Les stratégies de recherche d'une solution

On peut comparer le problème de la compréhension d'une phrase avec la recherche d'une solution dans l'espace des solutions c.à.d. des phrases possibles en fonction des contraintes posées par la structure du langage, le lexique et la chaîne de phonèmes représentant la phrase. Les règles régissant ce parcours constituent la stratégie de recherche. L'utilisation d'une stratégie de recherche peut aboutir dans tous les cas à la solution optimale. On dira alors que la stratégie est une stratégie totale.

D'autre part, en imposant une contrainte supplémentaire, p. ex. temporelle, on peut n'aboutir qu'à une solution possible, pas nécessairement optimale. Les stratégies de recherche tenant compte de telles contraintes sont appelées stratégies heuristiques. Analysons maintenant ces différentes stratégies de recherche.

3.3.1. Stratégies totales

Les systèmes utilisant ces stratégies de recherche, parcourent tout l'espace des possibilités et garantissent donc l'obtention d'une solution optimale. Cependant dès que cet espace de recherche devient volumineux, ce qui est presque toujours le cas pour les langages pseudo-naturels, le temps que met un tel système à trouver la solution optimale devient prohibitif. Ces stratégies ne sont donc pas du tout adaptées aux langages pseudo-naturels dans le cas de systèmes exigeant une réponse immédiate ou tout au moins rapide.

3.3.2. Stratégies heuristiques

Afin de tenir compte de cette contrainte temporelle, d'autres règles de recherche ont été définies. Les plus connues sont :

A) La stratégie du 'meilleur d'abord'

La caractéristique de cette stratégie est qu'on n'explore que les hypothèses ayant la plausibilité la plus élevée. Si l'exploration de cette hypothèse conduit à une impasse, on revient en arrière pour reprendre le traitement avec l'hypothèse suivante présentant la plausibilité la plus élevée. Du moment qu'on a trouvé une solution, on arrête les traitements sans que rien ne garantisse qu'on ait trouvé une solution optimale. A chaque fois qu'on décide d'explorer une hypothèse, on retient les autres hypothèses comme points de reprise éventuels.

C'est cette stratégie qui a été utilisée dans l'analyseur syntaxique de MYRTILLE II présenté dans la deuxième partie de ce mémoire. Les avantages de cette approche sont la simplicité de réalisation et la rapidité d'exécution. Par contre l'inconvénient majeur de cette méthode est le risque d'erreur élevé qui entraîne parfois un nombre de retours en arrière prohibitif.

B) La recherche en faisceau

Cette stratégie n'explore que les hypothèses les plus plausibles, c.à.d. les poursuit en parallèle et de façon synchronisée. C'est donc une stratégies des 'quelques meilleurs d'abord'. L'avantage en est que de cette manière on augmente la partie de l'espace explorée, ce qui en même temps implique un inconvénient car le parcours risque de devenir coûteux.

C) Le décodage séquentiel

C'est une stratégie de recherche où on parcourt simultanément plusieurs chemins possibles en parallèle. La limitation du nombre de chemins peut se faire de différentes manières. On peut même aller jusqu'à explorer toutes les solutions et se rapprocher ainsi des stratégies totales. Un avantage en est qu'on a à sa disposition un grand nombre de choix tandis que comme on mène plusieurs analyses simultanément (à la manière de la recherche en faisceau), le parcours risque de prendre beaucoup de temps.

D) La recherche par îlots de confiance

C'est la stratégie qu'on applique dans le cas d'une analyse du milieu vers les côtés, les autres n'étant valables que dans le cas d'une analyse gauche-droite. Elle consiste à s'appuyer sur un certain nombre de mots bien reconnus et d'essayer de faire concorder différentes interprétations partielles, ceci restant la difficulté principale.

Ayant ainsi présenté les différentes stratégies d'utilisation des informations linguistiques, nous pouvons passer aux différents modèles de représentation couramment utilisés.

4. MODELES DE REPRESENTATION DES LANGAGES

PSEUDO-NATURELS (1)

Nous arrivons maintenant au dernier chapitre de la partie consacrée aux connaissances générales nécessaires pour réaliser un système de reconnaissance automatique de la parole. Ce chapitre est cependant un peu plus spécialisé car il insiste surtout sur les modèles de représentation des langages pseudo-naturels, modèles généralement assez complexes à cause des informations sémantiques à prendre en compte. Les langages artificiels se basent en effet sur une description essentiellement syntaxique.

Dans ce chapitre comme dans le précédent, je respecte la classification proposée par J.M. Pierrel :

- Modèles purement syntaxiques
- Modèles syntaxico-sémantiques
- Modèles lexicaux et/ou sémantiques.

---

(1) op. cit. page 28 (2)

4.1. Les modèles purement syntaxiques

4.1.1. Les grammaires hors contexte de CHOMSKY

C'est le modèle le plus connu, qui permet de décrire à l'aide règles de production l'ensemble des phrases possibles d'un langage suivant un mécanisme génératif (1).

Dans ce genre de modèle on dispose d'un ensemble de 'terminaux', d'un ensemble de 'non-terminaux' d'un axiome et d'un certain nombre de règles de production. Les règles de production sont constituées en partie gauche d'un non-terminal et en partie droite d'une suite de terminaux et non-terminaux, la signification d'une telle règle étant la possibilité de substituer le non-terminal de gauche par la suite de droite. Une règle de production au moins doit avoir en partie gauche l'axiome. Aucune ne peut le contenir dans le membre droit. Chaque non-terminal doit apparaître au moins une fois dans le membre gauche d'une règle de production. L'application en cours de traitement de n'importe quelle règle de production doit toujours permettre d'aboutir finalement à une suite constituée exclusivement de terminaux. N'importe quelle suite de terminaux obtenue par ce mécanisme de substitution à partir de l'axiome est une phrase du langage (encore appelée expression bien formée).

Exemple :

Soit l'axiome A d'un langage quelconque,
soit l'ensemble des non-terminaux $\{a,b\}$,
soit l'ensemble des terminaux $\{c,d,e\}$,
et soient les règles de production suivantes :

$A \rightarrow a$
$a \rightarrow ab$
$a \rightarrow c$
$b \rightarrow ab$
$b \rightarrow de$

alors la suite 'cde' est une expression bien formée.

---

(1) voir Leroy H.
Cours de théorie des langages (2ème licence Info.)
Namur 1981-82

Ce modèle est parfaitement adapté à la représentation des langages artificiels. Il peut être utilisé pour la représentation des langages pseudo-naturels, mais les informations qu'il fournit doivent alors être complétées par des informations sémantiques. Les avantages en sont :

- Facilité d'emploi du modèle.
- Spécification précise des procédures d'analyse.
- Facilité de traitement automatique, obtention rapide d'une représentation de la structure de la phrase analysée.
- Indépendance de la définition de la structure vis-à-vis du vocabulaire.

Les désavantages en sont :

- Lourdeur, c.à.d. nombre élevé de règles de production et indétermination dans le cas de langages pseudo-naturels.
- Perte de temps due à l'utilisation d'un modèle génératif pour la reconnaissance.
- Trop grande importance de petits mots dont la reconnaissance est difficile et sur lesquels la structure est axée.
- Difficulté de prendre en compte des informations contextuelles.
- Lorsque pour les terminaux on ne descend qu'au niveau des classes grammaticales, trop peu d'informations sont fournies par ces terminaux.
- Séparation trop catégorique entre syntaxe et sémantique.

Les autres modèles qui se rapprochent tous de ce modèle Chomskyen sont les suivants :

4.1.2. Les matrices d'adjacence

Ce sont des matrices qu'on utilise pour décrire les automates finis (cfr. paragraphe suivant). Un élément de cette matrice indique dans quel état va tomber l'automate en fonction de l'état actuel de l'automate (qui détermine la ligne) et des conditions qui sont vérifiées (qui déterminent la colonne). Elle décrit donc les transitions possibles entre les différents états (1).

4.1.3. Les automates d'états finis

Un automate d'états finis est caractérisé par un certain nombre fini d'états et des transitions possibles entre ces états. L'automate se trouve dans un état et un seul à la fois. La transition d'un état vers un autre se fait lorsqu'une certaine condition est réalisée, en général lorsqu'un certain caractère ou un certain 'terminal' est présent à l'entrée (2).

---

(1) & (2) Leroy H.
Cours de théorie des langages (2ème licence Info.)
Namur 1981-82

4.1.4. Les grammaires transformationnelles

Elles permettent, par des manipulations de structure, d'obtenir différentes représentations de la structure d'une phrase, possédant toutefois la même signification. Il s'agit d'un ensemble de règles permettant de passer de la représentation de la structure profonde d'une phrase à la représentation de sa structure de surface et inversément.

4.2. Les modèles syntaxico-sémantiques

Les modèles purement syntaxiques se révélèrent très vite inefficaces pour le traitement automatique des langages pseudo-naturels. C'est pourquoi on a commencé à élaborer des modèles faisant intervenir autant que possible la sémantique en lien avec la syntaxe dans la représentation de la structure des langages naturels ou pseudo-naturels. Actuellement, les plus connus sont :

4.2.1. Les grammaires sémantiques

Ce modèle s'inspire très fortement des grammaires hors contexte de Chomsky. Il s'agit en fait d'une grammaire du même type, mais l'ensemble des terminaux n'est plus un ensemble de classes grammaticales mais un ensemble de classes sémantiques définies par le concepteur de manière à avoir un sens précis en fonction du contexte. Les avantages d'une telle grammaire sont :

- On obtient un lien étroit entre syntaxe et sémantique.

Les désavantages en sont :

- On obtient trop vite une dépendance de la structure du langage vis-à-vis de l'application envisagée.

A part cela, les grammaires sémantiques présentent les mêmes avantages et inconvénients que les grammaires hors contexte.

4.2.2. Les grammaires par cas

Ce modèle repose en grande partie sur l'idée que, pour comprendre une phrase, il n'est pas nécessaire de pouvoir établir l'arbre syntaxique complet de cette phrase. On essaie plutôt de retrouver les composants essentiels de la phrase. Pour cela on cherche à construire un 'frame', qui est un ensemble d'attributs, pour chaque verbe. Ce modèle bien adapté à une reconnaissance commençant par le verbe possède les inconvénients suivants :

- Trop grande importance des mots grammaticaux permettant de retrouver les arguments d'un verbe.

- Difficulté dans l'émission des hypothèses sur la position des différents mots. Ce modèle est donc mal adapté aux méthodes descendantes.
- Un nombre trop important de 'frames' à définir pour les verbes en français.

4.2.3. Les grammaires systémiques

Ce sont des modèles qui, à l'aide d'un graphe, font ressortir les différentes possibilités, c.à.d. les décisions à prendre, lors de l'analyse d'une phrase. Les inconvénients de ces modèles sont assez comparables à ceux des grammaires par cas. Le problème de ces modèles est que leur valeur théorique ne va pas nécessairement de pair avec leur efficacité qui dépend de procédures de traitement bien adaptées. Selon J.M. Pierrel, deux réalisations pratiques sont assez générales pour fonctionner avec différents modèles linguistiques :

- Les réseaux de transition de WOODS (1)
- Les procédures de WINOGRAD (2)

Une des contraintes principales dans le traitement de la parole est une contrainte temporelle; il s'agit donc - et c'est ce à quoi s'appliquent ces réalisations - de réduire à un minimum le temps perdu par des retours en arrière.

---

(1) Woods W. A.
Transition Network Grammars for natural language analysis
CACM, vol 13, n° 10, October 1970, p 591-602
art. cit. in op. cit. page 28 (2)

(2) Winograd T.
Understanding natural language
Academic press, New York 1972
op. cit. in op. cit. page 28 (2)

A) Les réseaux de transition de WOODS

Woods, dans cette réalisation, se base sur les grammaires hors contexte de Chomsky qui peuvent également être représentées par des réseaux et non seulement par des règles de production tel que nous l'avons vu dans le paragraphe 4.1.1.. Ces réseaux, Woods les élargit en associant à chaque arc du réseau une procédure permettant de construire la structure de la phrase et de prendre en compte le côté transformationnel des grammaires proposées par Chomsky. D'où le nom de A.T.N. (Augmented Transition Networks). Ces ATN sont composés de noeuds et d'arcs représentant les états et les transitions possibles. A chaque arc, correspondant à une transition, est associée une condition indiquant si cette transition est possible et si oui, un certain nombre d'actions à effectuer. Ces actions correspondent aux procédures dont on a parlé plus haut et ont pour but la construction de la structure de la phrase.

La définition des structures du langage dans le système MYRTILLE II s'inspire de ce genre de réalisation. Les principaux avantages des ATN sont :

- La prise en compte des transformations de façon dynamique lors de l'analyse.
- La possibilité d'analyser séparément différents fragments de la phrase grâce à l'utilisation de réseaux partiels, d'où la possibilité d'interrompre une analyse et de ne la reprendre que lorsqu'apparaissent les informations la concernant.
- L'adjonction de procédures aux arcs permettant la prise en compte de toutes sortes d'informations contextuelles.

Les inconvénients sont essentiellement les mêmes que ceux des modèles linguistiques dont il s'inspire : Les grammaires de Chomsky et les grammaires par cas. Même si Woods, afin de dépasser ces limites, utilise des ATN traitant une grammaire sémantique (dans le cas du système HWIM), ce système possède les mêmes désavantages que les grammaires sémantiques, c.à.d. une trop grande spécificité de la définition des structures du langage vis-à-vis de l'application choisie.

B) Les procédures de WINOGRAD

L'approche de Winograd ressemble à celle de Woods. Cependant il reproche à ce dernier l'indéterminisme inhérent à son modèle et la construction d'une structure dont on ne voit pas l'utilité.

Pour sa part Winograd, lors de l'analyse, ne propose jamais qu'une possibilité unique et ce n'est qu'en cas d'impasse qu'il revient en arrière dans son analyse en exploitant au maximum les renseignements déjà receuillis. De plus il ne construit pas d'arbre syntaxique mais ses commandes sont directement élaborées par un robot.

L'inconvénient essentiel de cette méthode est, comme pour les ATN sémantiques, la trop grande dépendance des procédures associées à chaque cas vis-à-vis de l'application choisie.

4.3. Les modèles lexicaux et/ou sémantiques

Contrairement à ce qui se passait dans les deux premiers types de modèles, on ne se base plus ici sur une définition de la structure du langage pour représenter les connaissances linguistiques nécessaires à la reconnaissance et la compréhension d'une phrase mais sur la définition des mots et de leur contexte. On peut distinguer deux types :

4.3.1. Modèles lexicaux sur base de la définition de transitions entre les mots

Dans ce genre de modèle on définit toutes les transitions possibles entre les différents mots du langage. Cette définition est à la fois une définition syntaxique et sémantique de ce langage. L'avantage en est la rapidité d'exécution grâce aux réseaux précompilés, tandis que l'inconvénient en est une forte dépendance du réseau vis-à-vis de l'application choisie.

4.3.2. Modèles lexicaux sur la base de la définition de fonctions liées aux mots

Dans ce type de modèle on ne décrit pas l'ensemble des phrases possibles comme c'était le cas pour les modèles se basant sur les transitions entre les mots mais on associe un certain nombre d'attributs syntaxiques et sémantiques à chaque mot.

Ces modèles sont en général très bien adaptés à l'interprétation d'une phrase mais ne donnent pas la moindre information sur la position des mots dans la phrase d'où des difficultés pour l'émission d'hypothèses.

Comme dans le système MYRTILLE II nous voulons reconnaître un langage pseudo-naturel dont la description dépasse les simples C-grammaires dont les terminaux sont les mots du langage, il faut introduire un certain nombre d'informations lexicales dans ce système. Ces informations ne doivent pas constituer une définition complète des connaissances sur le langage mais doivent compléter les informations fournies par d'autres sources contrairement aux deux modèles lexicaux précédents. Comme nous n'allons plus y revenir, signalons que les informations qui y sont prises en compte sont :

- Des informations syntaxiques.
- Des informations syntaxico-sémantiques.
- Des informations phonétiques et phonologiques.

Ayant maintenant terminé la partie sur les connaissances générales nous passons à l'exposé du système MYRTILLE II. Pour comprendre cet exposé, le lecteur doit bien connaître les sources d'informations dont dispose le système et qui ont été exposées dans le chapitre 2. De plus il est important de bien comprendre le fonctionnement des méthodes d'analyse mixtes et du traitement gauche droite du signal. Rappelons enfin que le modèle de représentation du langage utilisé dans le cadre de ce système est du type syntaxico-sémantique et plus particulièrement du type systémique, analogue aux ATN de Woods.

PARTIE II

MYRTILLE II est un système de compréhension automatique du discours reconnaissant un langage pseudo-naturel, sous-ensemble assez vaste du français parlé. Le modèle sur lequel se base la description de la srtucture de ce langage est du type syntaxico-sémantique, complété par un certain nombre d'informations lexicales (cfr. chapitre 4).

Après avoir examiné le fonctionnement général du système, nous étudierons en détail la définition des structures du lanage à l'aide des RNP (Réseaux à Noeuds Procéduraux) et nous traiterons de l'implémentation sur ordinateur des RNP et de l'analyseur chargé de l'exploitation des informations contenues dans ces RNP.

5. Description du système MYRTILLE II

Ce système utilisant des contraintes hiérarchisées de types morphologiques, syntaxiques et sémantiques (cfr. chapitre 2 : éléments d'un langage; et chapitre 3 : organisation interne d'un SRP) se divise en deux grands sous-systèmes (1) :

- Le système acoustico-phonétique
- Le système syntaxico-sémantique.

---

(1) Haton J.-P., Messenet G., Pierrel J.-M. & Sanchez C.
La chaîne de compréhension parlée du système MYRTILLE II
Actes du congrès de l'AFCET, tome 2, 13-15 novembre 1978, p 315-326

Pierrel J.-M.
MYRTILLE II: Un système de compréhension du discours continu
Exposé à l'école d'été de l'AFCET, Namur, Juillet 1978

Pierrel J.-M. & Haton J.-P.
Syntactic-semantic interpretation of sentences in MYRTILLE II speech understanding system
IEEE 4 - 1980

Pierrel J.-M.
Utilisation de contraintes linguistiques en compréhension automatique de la parole continue : le système MYRTILLE II
TSI, vol 1, n° 5, 1982, p 403-421

op. cit. page 28 (2)

Dans le module "segmentation" on découpe le signal vocal en segments minimaux et on étudie la stabilité du signal de la parole. Cela nous permet de déduire la position des zones stables et des zones transitoires. Les zones stables correspondent en général aux corps de voyelles

5.1.3. Le lissage de la chaîne phonémique

Il s'agit essentiellement d'optimiser les traitements au niveau linguistique, en supprimant ou diminuant une trop grande redondance dans la chaîne obtenue. Les critères utilisés sont de deux types :

- purement physiques : p. ex. suppression des segments inférieurs à o,1 sec.;
- morphologiques : concaténation de segments successifs de même type; suppression de queues de phonèmes non-significatives; etc.

Comme nous l'avons déjà indiqué (cfr. page 34), un des résultats du système acoustico-phonétique sera une transcription de la phrase prononcée en pseudo-chaîne codée en notation phonétique. Cette transcription, vu les limitations des algorithmes du système acoustique et vu les difficultés d'interpréter de mauvaises prononciations, comportera des erreurs de plusieurs types (cfr. page 30) :

- Erreurs de substitution :
  = confusions entre phonèmes assez proche;
  = altérations.
- Erreurs d'élision.
- Erreurs d'insertion.

Afin d'augmenter les chances de reconnaissance de la phrase, les concepteurs ont donné au système la possibilité de réponses multiples. Le nombre de réponses possibles a été fixé à 3, ce qui semble être un bon compromis entre la validité de l'information et la possibilité pour l'algorithme de recherche lexicale de travailler en temps réel.

---

(1) Markel J.D. & Gray A.H.
Linear prediction of speech
Springer Verlag 1976
op. cit. in
art. cit. page 55 (1)b

5.2. Le système syntaxico-sémantique

Pour comprendre la structure du système MYRTILLE II, il faut se rappeler qu'il fonctionne selon le principe d'hypothèses-tests (cfr. page 51), que la structure interne est de type pseudo-parallèle (cfr. page 56), que le modèle de représentation du langage pseudo-naturel utilisé est du type grammaire systémique et que comme nous sommes dans le cas de la reconnaissance d'un langage pseudo-naturel, différentes sources d'informations doivent être prises en compte.

5.2.1. L'architecture de MYRTILLE II

La structure générale de MYRTILLE II est décrite par la figure 5.1.. On y distingue les composants suivants :

- Le module de traitement acoustico-phonétique. C'est lui qui est chargé des traitements décrits dans la section précédente.
- Le module Départ, chargé de déterminer le point de départ des traitements syntaxico-phonétiques.
- Le module Aiguillage, qui, lui, est chargé de la gestion du pseudo-parallélisme (cfr. page 56).
- Les modules Parser, Prophon et Seman, chargés de la prise en compte des différentes informations linguistiques.
- Le module Gestion des hypothèses, dont la tâche est de gérer les hypothèses émises par les trois modules précédents.
- Le module Reconnaissance phonémique qui au vu des hypothèses doit faire des accès au lexique et à la chaîne phonémique.
- Le module Validation devant valider les hypothèses émises et enfin :
- Le module Sortie qui doit générer une réponse appropriée.

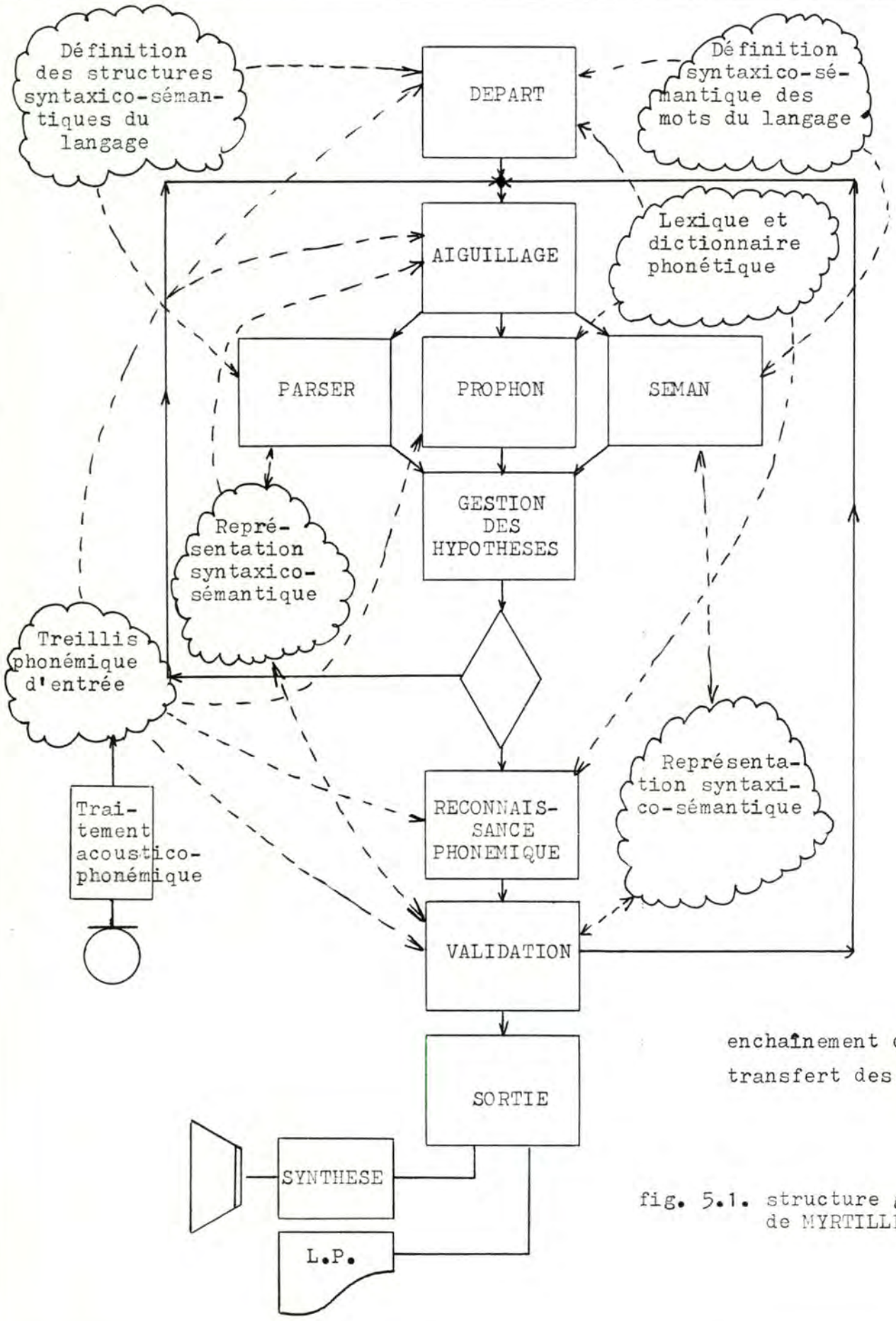


fig. 5.1. structure générale de MYRTILLE II

5.2.2. Fonctionnement de MYRTILLE II

Le système fonctionne essentiellement en quatre étapes :

- Initialisation
- Emission des hypothèses
- Validation des hypothèses
- Sortie

Examinons chacune de ces phases plus en détail.

A) Initialisation : module DEPART

Le rôle de ce module est de déterminer le point de départ du processus d'analyse de la phrase. Dans le cas d'une analyse gauche-droite, son rôle est quasi inexistant; dans le cas d'une analyse du milieu vers les côtés, c'est lui qui détermine l'îlot de confiance (cfr. pages 53 et 60) sur lequel s'appuie l'ensemble des traitements. Le traitement gauche-droite de la phrase peut être inadapté lorsque le début de la phrase est fortement entâché de bruits et diificile à retrouver. Dans ce cas, la compréhension doit pouvoir démarrer au milieu. Le modèle de représentation des structures syntaxico-sémantiques du langage choisi dans le cas de MYRTILLE II, c.à.d. les RNP (pour une description détaillée, voir chapitre suivant), permet très bien ce genre de traitements.

B) L'émission des hypothèses

Le principe général en est le suivant :

- au début, l'ensemble des hypothèses (sur les mots constituant la phrase) correspond à l'ensemble des mots du dictionnaire de l'application;
- la prise en compte des différentes informations (définition des structures syntaxico-sémantiques du langage, définition syntaxico-sémantique des mots, traits prosodiques et phonologiques etc) permet de restreindre ces hypothèses et de déterminer un sous-ensemble de mots qui seront testés puis validés au niveau de la reconnaissance phonémique par un module de recherche lexicale.

Cette phase est très importante car elle permet de réduire sensiblement le temps de traitement, proportionnel au nombre de mots à tester, en diminuant fortement l'indéterminisme sur les mots; indéterminisme dû aux limites de la reconnaissance phonémique travaillant sur un treillis de phonèmes fortement entâché d'erreurs.

Examinons les principaux modules mis en oeuvre lors de cette phase de traitement.

a) PARSER

Son rôle :

- Restriction des hypothèses en tenant compte de la définition syntaxico-sémantique des structures du langage ainsi que du contexte de la phrase traitée.
- Emission d'hypothèses :
  = Au niveau des procédures internes du réseau syntaxico-sémantique lorsque, compte tenu du contexte déjà traité, il sélectionne une ou plusieurs sorties de ces procédures.
  = Au niveau des branches linéaires lorsqu'il rencontre un terminal de la grammaire.

Ses caractéristiques :

- Cette procédure est récursive.
- Elle autorise les retours en arrière dans le cas d'une impasse.
- Enfin elle peut être interrompue à chaque pas pour permettre la prise en compte des autres sources d'informations et elle doit pouvoir reprendre à un autre endroit que celui où elle est interrompue à cause des fortes interactions avec les autres modules.

Les techniques utilisées sont très différentes de celles utilisées traditionnellement en analyse syntaxique. Il s'agit essentiellement d'un parcours du RNP. Deux types de parcours sont envisagés :

- Parcours de type descendant gauche-droite (cfr. pages 50 et 52).
- Parcours de type ascendant et du milieu vers les côtés (cfr. pages 49 et 53).

b) SEMAN

On prend ici en compte la définition syntaxico-sémantique des mots. Ce module restreint les hypothèses en fonction des dépendances conceptuelles acceptées par la partie de la phrase déjà traitée mais aussi en fonction de la structure déjà reconnue.

Il sélectionne, dans l'ensemble des hypothèses, les différentes sous-classes grammaticales et entités possibles, compte tenu du contexte déjà traité.

De plus, en cas d'échec d'une phase de reconnaissance guidée par la syntaxe, SEMAN fournit un point de reprise au module PARSER pour poursuivre un traitement du milieu vers les côtés, en déterminant le mot dominant les grands composants de la phrase non encore reconnus. Il faut encore remarquer que ce module n'est pas le seul à prendre en compte des informations sémantiques. En effet, les procédures du RNP activées par le module PARSER travaillent aussi sur ces informations pour sélectionner les différentes sorties possibles.

c) PROPHON

Son rôle : Pondérer les hypothèses au niveau des mots en fonction des traits prosodiques et de la structure phonémique du contexte à reconnaître.

Les principaux tests utilisés sont :

- La longueur phonémique par la prise en compte des pauses et des marqueurs de segmentation prosodique.
- Les patrons intonatifs.
- La présence ou non de plosives ou fricatives.

Remarquons ici aussi que le module PROPHON n'est pas le seul à travailler sur les informations prosodiques ou phonémiques. Ce sont encore une fois les procédures associées aux noeuds du RNP qui exploitent également ces informations.

Dans l'architecture de MYRTILLE II qui est organisée de manière pseudo-parallèle (cfr. page 56), les trois modules se placent sur le même plan. C'est le module AIGUILLAGE qui est chargé de la supervision de ces trois procédures. Il les active et détermine l'ordre d'activation, en tenant compte du contexte déjà traité et des objectifs recherchés.

La compatibilité des résultats des différents modules est assurée par le module GESTION DES HYPOTHESES.

C) La validation des hypothèses

Elle se décompose en deux traitements :

a) La reconnaissance phonémique

Son rôle : Valider ou rejeter les hypothèses émises par les niveaux sémantiques, syntaxiques ou prosodiques par l'affectation d'un score de reconnaissance à chaque hypothèse. Celles dont le score est inférieur à un certain seuil sont éliminées, les autres classées dans l'ordre croissant des scores.

Cette reconnaissance phonémique se fait elle-même en deux temps :

- recherche et reconnaissance lexicale du radical du mot;
- traitement de la partie désinence du mot par la procédure d'altération phonologique liée à ce mot.

Cela est dû au fait que, pour certains mots, on ne conserve pas toujours la transcription phonétique entière dans le lexique, mais uniquement la racine. Pour la partie restante du mot on ne conserve que des informations sur des règles d'altération phonologique liées à ce mot.

b) La validation proprement dite

Son rôle est de sélectionner une hypothèse et de mettre à jour la représentation syntaxico-sémantique de la partie d'énoncé déjà traitée. La méthode utilisée actuellement est celle de la 'meilleure d'abord', c.à.d. celle dont le score est le plus proche de 1.

De plus, elle détermine les points de reprise des modules AIGUILLAGE, PARSER, SEMAN et PROPHON lors de retours en arrière éventuels.

Enfin, elle doit calculer un score de reconnaissance cumulé pour la partie déjà traitée et décider de la fin du processus de reconnaissance.

D) Sortie

Pour l'instant, la sortie est encore assez primitive. Il s'agit simplement de l'écriture de la phrase reconnue et de sa représentation syntaxico-sémantique.

Il est cependant envisagé de remplacer cette sortie par une synthèse de la réponse correspondant à la demande formulée. Pour cela il faudra encore ajouter au système de traitement de la parole un système général de question-réponse capable de provoquer cette action.

Ayant décrit la structure générale du système et mis en évidence les différents modules qui interviennent dans le traitement au niveau linguistique, abordons la description des connaissances sur la structure du langage.

6. DESCRIPTION DES STRUCTURES SYNTAXICO-SEMANTIQUES DU LANGAGE DANS MYRTILLE II

Le but de ce chapitre est d'expliquer le modèle de représentation des structures du langage traité par le système MYRTILLE II et ce en partant des objectifs principaux et des choix de réalisations que se sont fixés les concepteurs du système (1).

---

(1) arts. et op. cit page 75 (1)

6.1. Objectifs et choix de réalisation

6.1.1. Les objectifs principaux

Les objectifs peuvent être résumés comme suit :

- Indépendance de la structure vis-à-vis de l'application.
- Structure décrivant un large sous-ensemble du français parlé.
- Structure porteuse d'informations positionnelles sur les mots.
- Structure permettant de construire en cours de reconnaissance un arbre syntaxico-sémantique de la partie déjà traitée.
- Structure adaptée au traitement de la parole continue :
  = intégrant des traitements particuliers pour les mots courts de liaison;
  = gérant le mieux possible l'indéterminisme dû à la définition syntaxique.

6.1.2. Les choix de réalisation

D'où les choix de réalisation suivants :

- Modèle de type grammaire hors contexte (cfr. page 62).
- Limitation de la description syntaxique aux classes grammaticales classiques (verbes, adjectifs etc).
- Adjonction de procédures de traitement syntaxico-sémantique afin de permettre des restrictions syntaxiques locales, traitant les mots courts de liaison et gérant l'indéterminisme inhérent à la description syntaxico-sémantique.

Le modèle ainsi obtenu correspond à un réseau de transition représentant une grammaire hors contexte à terminaux de type classes syntaxiques auquel on a ajouté une procédure à chaque noeud. Ce modèle a été appelé 'Réseau syntaxico-sémantique à noeuds procéduraux' ou RNP.

6.2. Définition des RNP

Un réseau à noeuds procéduraux se présente sous forme d'un ensemble de noeuds reliés entre eux par des branches linéaires. Dans ce réseau interviennent des informations syntaxiques et sémantiques mais aussi prosodiques et phonétiques. A chacun des noeuds est associée une procédure. L'entrée dans un réseau s'effectue par l'activation d'une procédure unique appelée 'procédure d'entrée', la sortie par l'activation d'une procédure également unique appelée 'procédure de sortie'. La figure 6.1. donne un exemple d'un tel réseau à noeuds procéduraux.

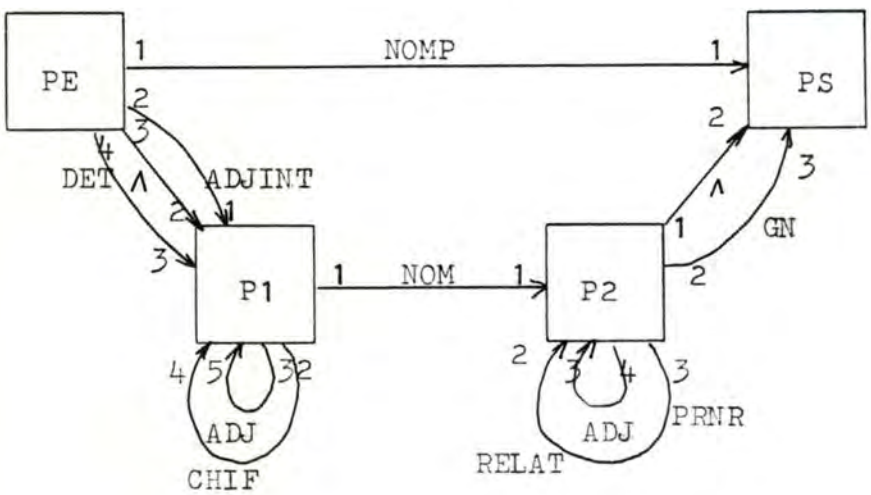


fig. 6.1. Exemple de RNP : Le réseau GN (groupe nominatif) de MYRTILLE II

Analysons ces différents composants.

6.2.1. Les noeuds procéduraux

Ils correspondent aux procédures associées à chaque noeud du réseau. Ils sont entièrement définis par :

- leur identifiant Px;
- le nombre d'entrées En;
- le nombre de sorties Es.

Par définition :

- pour les procédures d'entrée : En = 0;
- pour les procédures de sortie : Es = 0;
- pour les autres procédures : En $\geqslant$ 1 et Es $\geqslant$ 1.

Leur rôle est :

- rendre compte des phénomènes contextuels;
- traiter les mots courts de liaison;
- réduire l'indéterminisme apparaissant lors du parcours d'un tel réseau.

6.2.2. Les branches linéaires

Elles sont définies comme une suite d'arcs du réseau correspondant soit à des mots du lexique soit à des références à un autre sous-réseau, cette structure pouvant être récursive.

Les informations qu'elles fournissent sont essentiellement les mêmes que peut fournir une matrice d'adjacence (cfr. page 64).

6.3. Traitements associés aux RNP

Quels sont les traitements qu'on peut effectuer sur ces RNP? Avant de pouvoir exploiter des informations en machine, il faut les avoir stockées quelque part en machine. Il faut donc passer d'une représentation externe sous forme de graphique, comme le montre la figure 6.1. (cfr. page 88), vers une représentation interne sous forme de tableau composé d'un ensemble de pointeurs et d'autres éléments. Ce sera le rôle de la procédure de création d'un RNP. La procédure de parcours d'un RNP, quant à elle, aura pour but l'exploitation des informations contenues dans le RNP en vue de la reconnaissance d'une phrase. Il s'agira donc fondamentalement d'un analyseur syntaxique.

Dans les chapitres suivants nous allons analyser ces opérations qu'on peut effectuer sur les RNP.

7. PROCEDURE DE CREATION D'UN RNP

(programme CRERES (1))

7.1. Le rôle de cette procédure

Le rôle de la procédure de création d'un RNP est de permettre à un utilisateur non-informaticien de passer du RNP sous forme graphique à sa représentation en machine et de la mémoriser sur un fichier. Ce passage se fait à l'aide d'un dialogue homme-machine sur un terminal vidéo.

7.2. Les spécifications de cette procédure

Le problème est entièrement spécifié lorsqu'on connaît les éléments suivants :

- la forme de la représentation interne des RNP
  (pour la description détaillée de cette représentation interne, se référer à l'ANNEXE I, chapitre : Spécification du programme, section : Représentation interne des RNP);
- le dialogue de saisie des données
  (pour la description détaillée de ce dialogue, se référer à l'ANNEXE I, chapitre : Spécification du programme, section : Dialogue de saisie des données);
- de plus il faut respecter le fait que l'utilisateur peut être non-informaticien, d'où la nécessité de nombreuses vérification, de messages d'aide etc.
  (pour la liste détaillée des conditions imposées, se référer à l'ANNEXE I, chapitre : Spécification du programme, section : Compléments de spécification).

---

(1) Ce programme a été mis au point par l'auteur lors d'un stage effectué à Nancy durant le semestre d'hiver de l'année académique 82/83.

7.3. L'implémentation de cette procédure

Disons simplement ici que la procédure a été implémentée en PASCAL standard, que la structure fondamentale respecte autant que possible la structure de l'algorithme de J.-M. Pierrel écrit en FORTRAN. Tenant compte du fait que la procédure doit pouvoir être utilisée par un non-informaticien, de nombreuses procédures de vérification, de messages d'aide et de reprise des traitements en cas d'erreur ont été ajoutées. On trouvera la structure de l'algorithme FORTRAN ainsi que la description détaillée des traitements de CRERES dans le dossier d'analyse de laprocédure présenté en ANNEXE I. Le listing ainsi qu'un exemple de résultat est présenté en ANNEXE II. Le mode d'utilisation se trouve en ANNEXE III.

8. PROCEDURE DE PARCOURS D'UN RNP

(programme ANSYNT (1))

8.1. Le rôle de cette procédure

Le rôle de cette procédure de parcours d'un RNP est de parcourir pas à pas le réseau et d'activer, entre chaque pas de l'analyse des procédures de test, des hypothèses éventuellement émises (sur les mots à retrouver dans la phrase). De plus il s'agit de créer et de gérer une structure de liste parenthésée représentant la structure syntaxico-sémantique de la phrase analysée (p. ex. sous forme de pile).

8.2. Les spécifications de la procédure

La procédure est spécifiée si nous disons que nous voulons construire un analyseur syntaxique s'appuyant sur la définition d'un RNP comparable à ceux utilisés avec une C-grammaire, travaillant de manière descendante, traitant le signal acoustique (c.à.d. sa transcription phonétique) de gauche à droite (cfr. page 52) et utilisant une stratégie du meilleur d'abord (cfr. page 59). De plus les procédures de tests phonétiques et d'accès au lexique sont entièrement simulées par un dialogue via le terminal vidéo.

---

(1) Ce programme a été mis au point par l'auteur lors d'un stage effectué à Nancy durant le semestre d'hiver de l'année académique 82/83.

8.3. L'implémentation de la procédure

La procédure de parcours du RNP a également été implémentée en PASCAL. Quatre versions de cet analyseur sont proposées selon que :

- La définition des procédures associées aux noeuds procéduraux est incluse dans le code de l'analyseur ou non.
- L'indéterminisme rencontré au niveau de la définition syntaxico-sémantique est simple ou double.
  Dans le cas du simple indéterminisme, il se retrouve uniquement au niveau de la sortie des différents noeuds procéduraux.
  Dans le cas du double indéterminisme, on le retrouve non seulement au niveau des sorties des noeuds mais également au niveau de la reconnaissance d'un terminal lors du parcours d'un arc d'une branche linéaire, donc lors d'un accès au lexique.

Les détails de cette implémentation sont présentés en ANNEXE IV pour la partie analyse et en ANNEXE V pour la partie programmation.

9. CONCLUSION

Tentons de tirer un certain nombre de conclusions et notons quelques remarques générales sur les systèmes de reconnaissance de la parole.

Dans une première partie j'ai essayé d'exposer un certain nombre de concepts nécessaires à la compréhension des systèmes de reconnaissance automatique du discours continu. Après avoir défini ce qu'était un SRP, j'ai donné un aperçu historique sur les réalisations principales dans le domaine. J'ai ensuite essayé d'en dégager les principaux buts poursuivis, les avantages et les inconvénients de ces techniques. Le chapitre suivant, qui à mon avis est un des plus importants, décrit les connaissances fondamentales nécessaires à la réalisation pratique d'un SRP. Sans une maîtrise, au moins partielle, de ces disciplines, aucun traitement automatique de haut niveau de la parole ne serait possible. La définition de ces connaissances est ensuite suivie de l'explication des principales techniques d'utilisation de ces connaissances et de la description des modèles de représentation d'une partie de ces connaissances sur les lanages pseudo-naturels, aspect qui nous intéresse particulièrement dans le cadre de ce mémoire. Cet aspect concerne la définition des structures syntaxico-sémantiques des langages pseudo-naturels.

Après cette première partie où les concepts généraux ont été ainsi expliqués, nous avons abordé l'étude d'un cas pratique : le système MYRTILLE II. L'examen de l'architecture et du fonctionnement général du système nous ont permis d'aborder la description plus détaillée du modèle de représentation des structures syntaxico-sémantiques utilisé dans ce système ainsi que la description des traitements s'effectuant sur ces structures.

De tout ce qui a été dit, nous pouvons déduire que l'aspect syntaxique des langages est assez bien maîtrisé. Cela est dû principalement aux progrès spectaculaires en la matière auxquels on a assisté depuis le début du siècle. Ce qui pose encore beaucoup de problèmes aujourd'hui, c'est l'aspect sémantique des langages et les interactions de ce dernier avec la syntaxe. Ici, beaucoup de recherches fondamentales en linguistique sont encore nécessaires avant qu'on ne puisse enregistrer des progrès sensibles en compréhension automatique du discours continu.

La conséquence de cet état des choses est qu'aujourd'hui il est peu raisonnable de vouloir construire des systèmes reconnaissant tout. Vu nos connaissances actuelles sur le langage, ce genre de projet est voué à l'échec dès le début. Les projets qui ont le plus de chance de réussite pour l'instant sont des reconnaisseurs de mots isolés ou des reconnaisseurs du discours continu se basant sur des techniques fiables et bien maîtrisées. A partir de ces systèmes on pourra essayer de faire évoluer un de leurs aspects (p. ex. étendre la description syntaxico-sémantique ou lexicale du système) en restant toujours dans un contexte très restreint. C'est probablement de cette manière qu'on fera avancer les recherches.

ANNEXES

A N N E X E I

Dossier d'analyse du programme CRERES

Avant d'aborder le dossier d'analyse du programme CRERES, définissons encore les concepts de

- Terminaux généraux.
- Terminaux lexiques figés.
- Terminaux vrais.

Ce sont là les seuls types de terminaux pouvant apparaître dans la définition d'un RNP. On distingue dans le lexique entre :

- Terminaux indépendants de l'application, classe qui regroupe :
  - = Les terminaux vrais qui désignent l'ensemble des mots courts de liaison apparaissant de manière explicite dans le RNP.
  - = Les terminaux lexiques figés qui désignent l'ensemble des mots correspondant à des sous-classes grammaticales.
- Terminaux dépendants de l'application, on y retrouve
  - = Les terminaux généraux, l'ensemble des noms, verbes et adjectifs.

DOSSIER D'ANALYSE DU PROGRAMME

CRERES

0. TABLE DES MATIERES

0. TABLE DES MATIERES 0-1
1. SPECIFICATION DU PROGRAMME 1-1
1.1. REPRESENTATION INTERNE DES RNP 1-1
1.1.1. La représentation des noeuds procéduraux 1-1
1.1.2. La représentation des branches linéaires 1-2
1.1.3. La représentation du début et de la fin des sous-réseaux 1-2
1.1.4. Remarque 1-3
1.1.5. Exemple 1-3
1.2. DIALOGUE DE SAISIE DES DONNEES 1-4
1.3. COMPLEMENTS DE SPECIFICATION 1-6
2. ANALYSE DE L'ALGORITHME 2-1
2.1. STRUCTURE FONDAMENTALE 2-1
2.2. DESCRIPTION DETAILLEE DES TRAITEMENTS 2-15
2.2.1. Initialisation des valeurs de base du réseau 2-15
2.2.2. Modification des valeurs de base du réseau 2-15
2.2.3. Initialisation des valeurs de base d'un s-réseau 2-16
2.2.4. Traitement des noeuds procéduraux d'un s-réseau 2-17
2.2.5. Traitement des branches linéaires d'un s-réseau 2-18
2.2.6. Affichage du réseau 2-19
2.2.7. Enregistrement du réseau sur un fichier 2-19
3. IMPLEMENTATION 3-1
3.1. LE PROGRAMME PRINCIPAL 3-1
3.1.1. Initialisation 3-1
3.1.2. Traitement du réseau 3-2
3.1.3. Mémorisation du réseau 3-2
3.2. LES PROCEDURES D'INITIALISATION 3-3
3.2.1. INITNT 3-3
3.2.2. INITTV 3-3
3.2.3. INITTL 3-3
3.2.4. INITTG 3-3
3.2.5. MODINI 3-4
3.3. LES PROCEDURES DE TRAITEMENT DES SOUS-RESEAUX 3-5
3.3.1. TRDSR 3-5
3.3.2. TRNO 3-5
3.3.3. TRBR 3-5
3.4. LES PROCEDURES DE VERIFICATION DE COHERENCE 3-7
3.4.1. VERNO 3-7
3.4.2. VERBR 3-7
3.4.3. VERRE 3-7
3.5. LES PROCEDURES DE GESTION DE REPRISE 3-8
3.5.1. REPNO 3-8
3.5.2. REPBR 3-8

3.6. LES PROCEDURES HELP ET D'AFFICHAGE DE MESSAGES 3-9
3.7. LES PROCEDURES UTILITAIRES 3-9
3.7.1. ININT 3-9
3.7.2. INOUI 3-9
3.7.3. LECTURE 3-9
3.7.4. RECH 3-10
3.7.5. INSTRING 3-10
3.7.6. INARC 3-10
3.8. LES PROCEDURES D'AFFICHAGE ET DE MEMORISATION DU RESEAU 3-11
3.8.1. IMPRE 3-11
3.8.2. ECRRE 3-11
4. LISTING SOURCE 4-1

1. SPECIFICATION DU PROGRAMME

Le programme CRERES est une procédure de création et de modification d'un réseau à noeuds procéduraux. Sa spécification est donnée à la page 178 de la thèse d'état de Jean-Marie PIERREL sur le sujet : Etude et mise en oeuvre de contraintes linguistiques en compréhension automatique du discours continu. L'énoncé est le suivant :

" La représentation interne d'un réseau à noeuds procéduraux est assez complexe, or, un des avantages des R.N.P. réside dans le fait qu'étant visuels, ils sont faciles à comprendre et à définir. Il était donc normal de vouloir automatiser le passage de la représentation externe à la représentation interne. Pour cela, nous avons mis en oeuvre une procédure CRERES, qui crée la représentation interne à partir de données introduites par un programme conversationnel. Un tel programme peut être entièrement spécifié par la représentation interne déjà fournie et le dialogue mis en oeuvre."

Pour donner une spécification complète du programme, il suffit donc de rappeler la représentation interne d'un R.N.P. ainsi que le dialogue nécessaire pour obtenir les informations sur la représentation externe du R.N.P.

1.1. REPRESENTATION INTERNE DES R.N.P.

Nous rappelons brièvement la structure de la représentation interne des R.N.P., telle qu'elle est présentée à la page 176 de la thèse pré-citée. On y distingue :

1.1.1. La représentation des noeuds procéduraux

(| liste d'entrée | NOM | liste de sortie |)*

dans laquelle :

- liste d'entrée : correspond à une liste de pointeurs vers la fin des 'branches linéaires' arrivant à ce noeud; il y a autant de pointeurs que d'entrées possibles dans la procédure.

- NOM : codification du numéro de la procédure par un entier 0.

- liste de sortie : correspond à une liste de pointeurs vers le début des 'branches linéaires' issues de ce noeud; il y a autant de pointeurs que de sorties possibles de la procédure.

1.1.2. La représentation des branches linéaires

(| - PS | liste des noms | - PE |)*

dans laquelle :

- PS : est un pointeur vers une sortie d'un noeud procédural.

- liste des noms : donne de façon séquentielle et ordonnée la liste des non-terminaux formant la 'branche linéaire'.

- PE : est un pointeur vers une entrée d'un noeud procédural.

1.1.3. La représentation du début et de la fin des réseaux

- DEB(NTER) : élément d'un tableau pointant vers le noeud correspondant à la procédure d'entrée du réseau qui décrit le non-terminal NTER.

- FIN(NTER) : élément d'un tableau pointant vers le noeud correspondant à la procédure de sortie du réseau qui décrit le non-terminal NTER.

1.1.4. Remarque

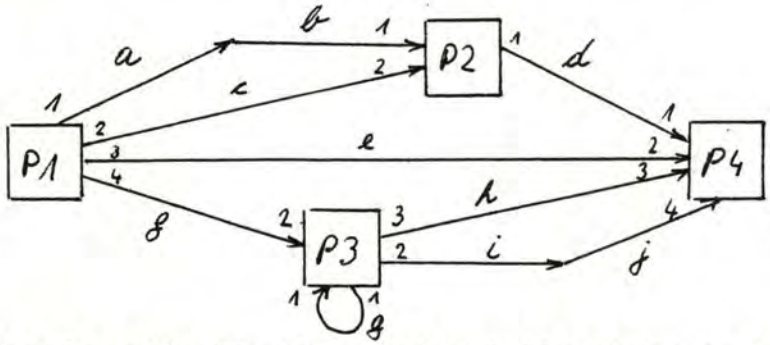
Une codification particulière doit permettre de distinguer dans les 'branches linéaires' :

- les mon-terminaux
- les accès au lexique
- l'élément vide

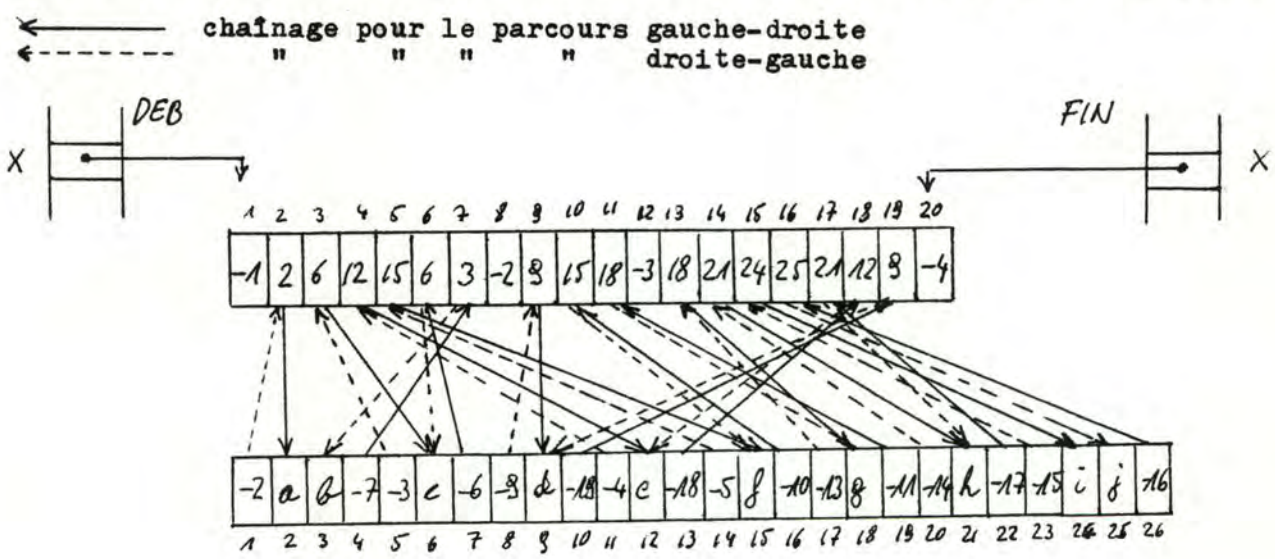
1.1.5. Exemple

Le passage de la représentation externe d'un R.N.P. à sa représentation interne est illustré dans l'exemple suivant :

- Soit le réseau de la page 160 de l'ouvrage pré-cité :



- La figure suivante schématise la représentation interne de ce réseau :



1.2. DIALOGUE DE SAISIE DES DONNEES

Nous donnons à présent un exemple de dialogue entre le programme et l'utilisateur nécessaire pour passer de la représentation externe à la représentation interne d'un réseau. Le dialogue se rapporte à l'exemple de réseau de la page 1-3 que nous appellerons RES1 :

(Les demandes du programme sont soulignées.)

Nombre de non-terminaux : 3
Ensemble de nom-terminaux : RES1 RES2 RES3
Nombre de terminaux vrais : 3
Ensemble de terminaux vrais : a b c
Nombre de terminaux à lexique figés : 3
Ensemble de terminaux à lexique figés : d e f
Nombre de terminaux généraux : 4
Ensemble de terminaux généraux : g h i j

Description du sous-réseau : RES1
Procédure d'entrée : P1
Procédure de sortie : P4
Nombre de procédures : 4

Description du noeud : P1
Nombre d'entrées : 0
Nombre de sorties : 4
Description du noeud : P2
Nombre d'entrées : 2
Nombre de sorties : 1
Description du noeud : P3
Nombre d'entrées : 2
Nombre de sorties : 3
Description du noeud : P4
Nombre d'entrées : 4
Nombre de sorties : 0

Sortie 1 du noeud P1 vers entrée : 1
Du noeud : P2
En passant par les arcs : a b
Sortie 2 du noeud P1 vers entrée : 2
Du noeud : P2
En passant par les arcs : c
Sortie 3 du noeud P1 vers entrée : 2
Du noeud : P4
En passant par les arcs : e
Sortie 4 du noeud P1 vers entrée : 2
Du noeud : P3
En passant par les arcs : f
Sortie 1 du noeud P2 vers entrée : 1
Du noeud : P4
En passant par les arcs : d
Sortie 1 du noeud P3 vers entrée : 1
Du noeud : P3
En passant par les arcs : g
Sortie 2 du noeud P3 vers entrée : 4
Du noeud : P4
En passant par les arcs : i j
Sortie 3 du noeud P3 vers entrée : 3
Du noeud : P4
En passant par les arcs : h

Description du sous-réseau : RES2

1.3. COMPLEMENTS DE SPECIFICATION

Le programme CRERES est un programme orienté utilisateur, càd qu'il devra observer certaines contraintes :

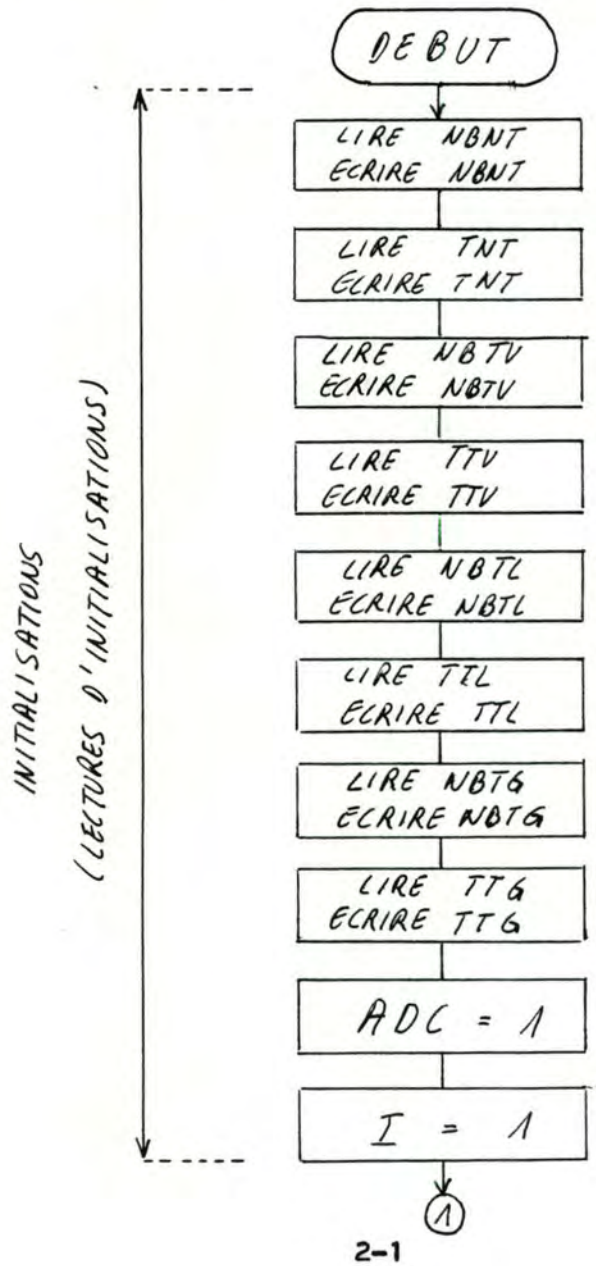
- ne pas imposer à l'utilisateur des formats de données fixes,
- aider l'utilisateur dans la mesure du possible, càd pouvoir répondre à des messages 'help' et permettre un certain nombre de retours en arrière ainsi que des vérifications visuelles,
- enfin le programme devra vérifier au maximum la cohérence des données introduites par l'utilisateur et signaler des erreurs éventuelles.

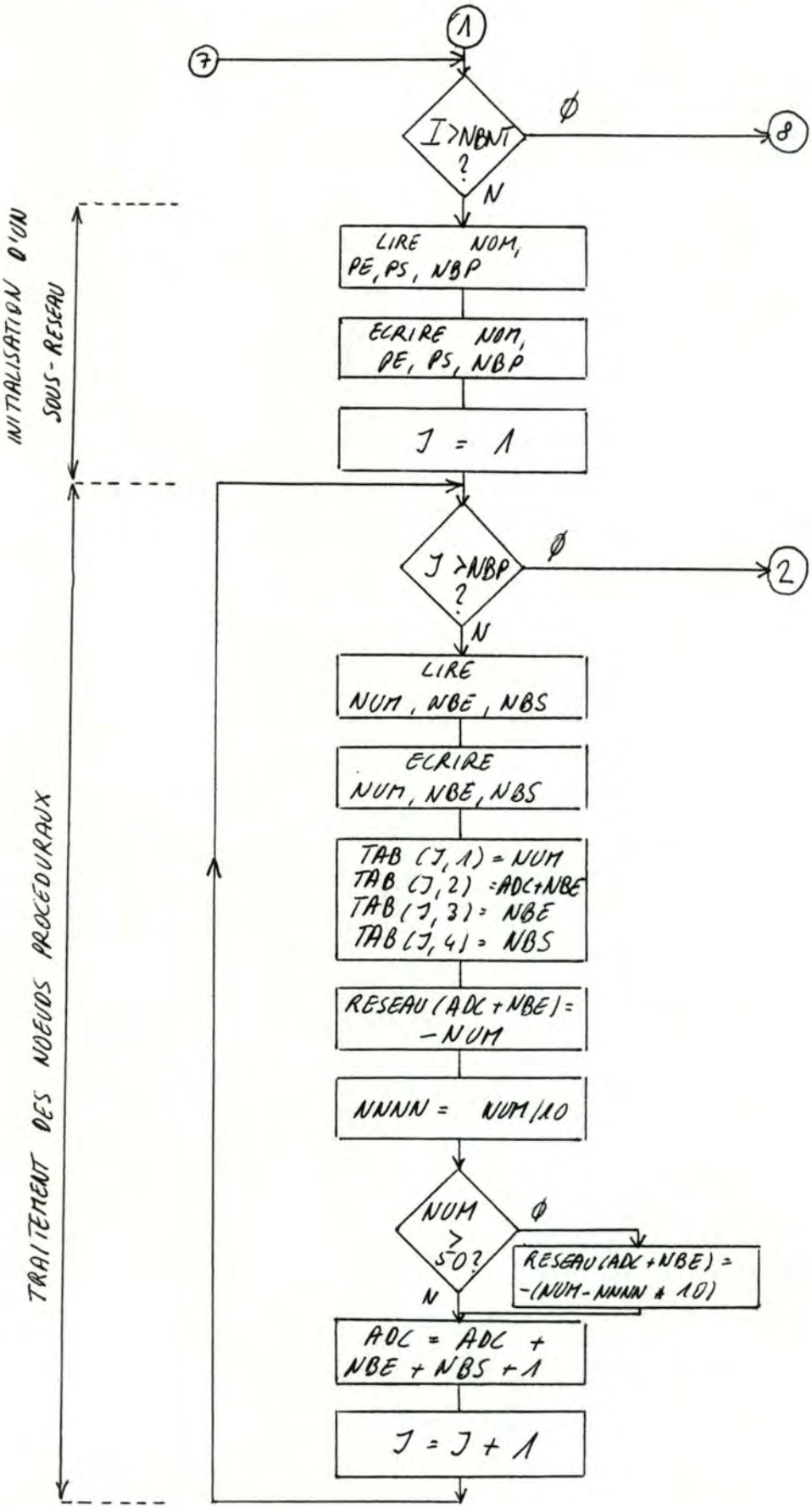
Pour modifier un réseau déjà existant, il suffira de le recréer par le programme CRERES. La modification d'un sous-réseau entraîne donc la réintroduction de tout le réseau.

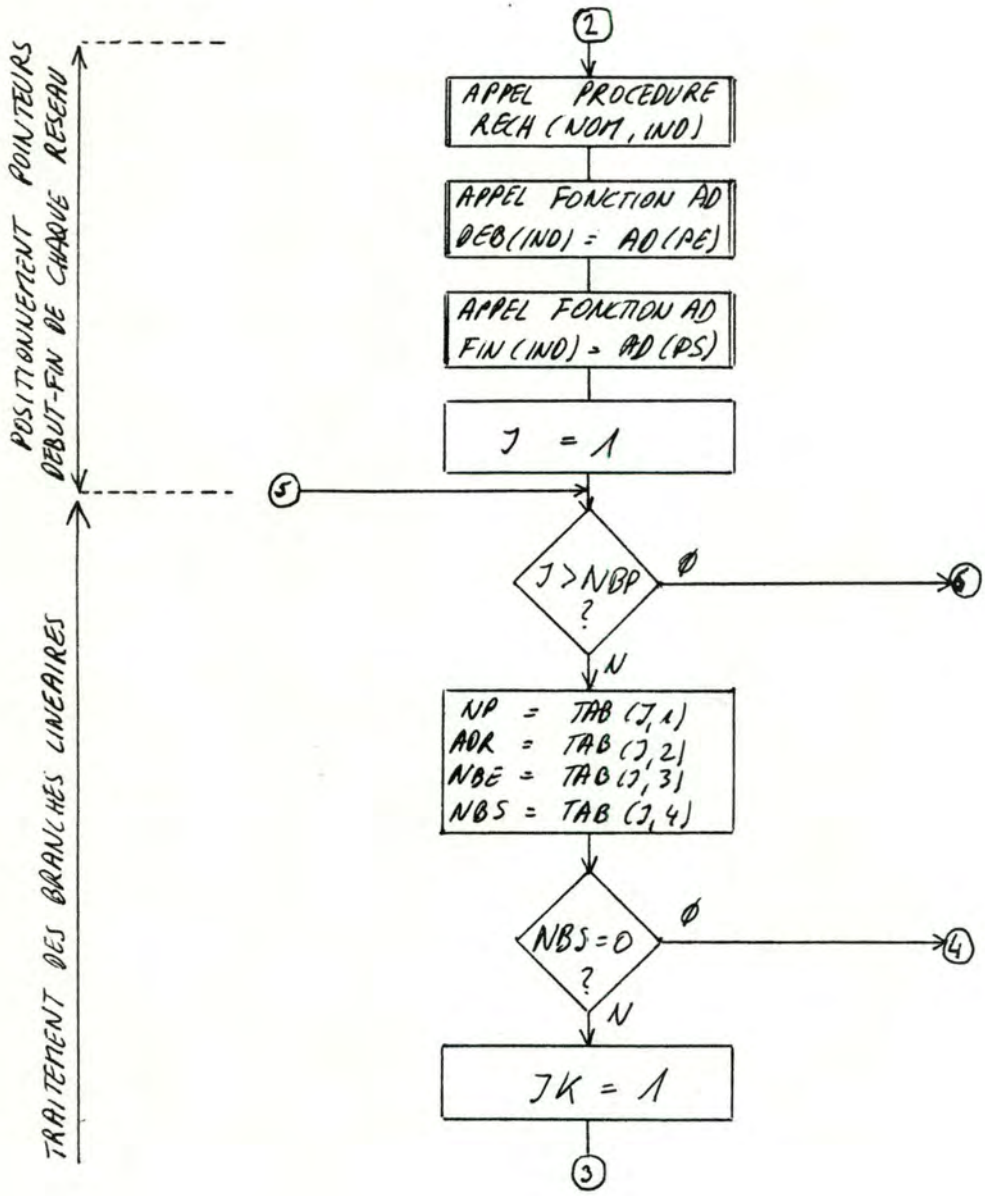
2. ANALYSE DE L'ALGORITHME

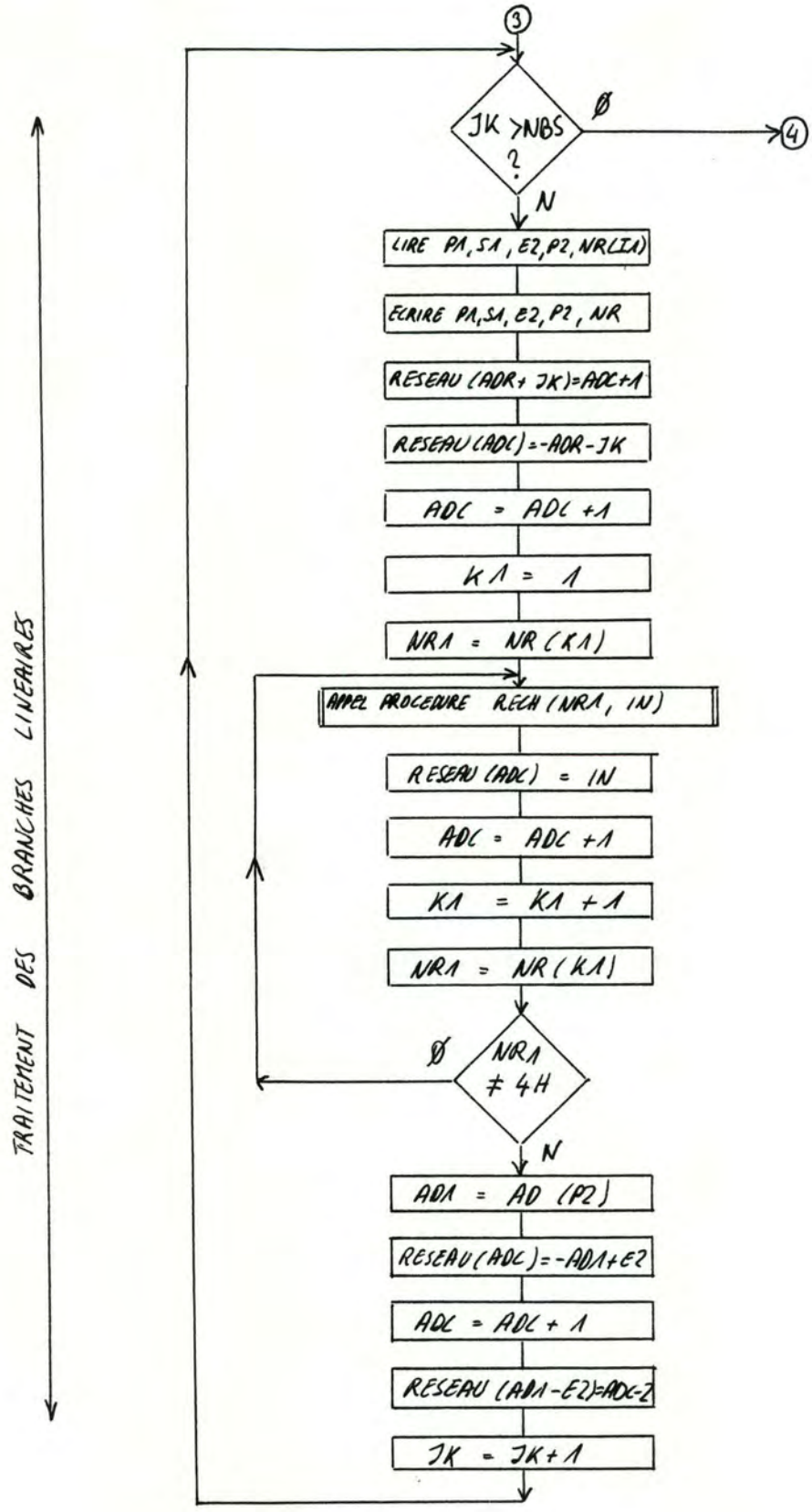
2.1. STRUCTURE FONDAMENTALE

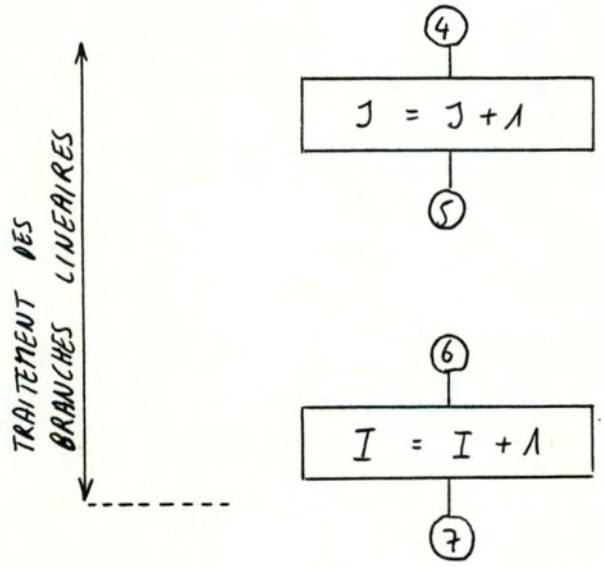
La structure fondamentale de l'algorithme s'imspire fortement du programme FORTRAN écrit par Jean-marie PIERREL dont l'ordinogramme est donné ci-dessous :

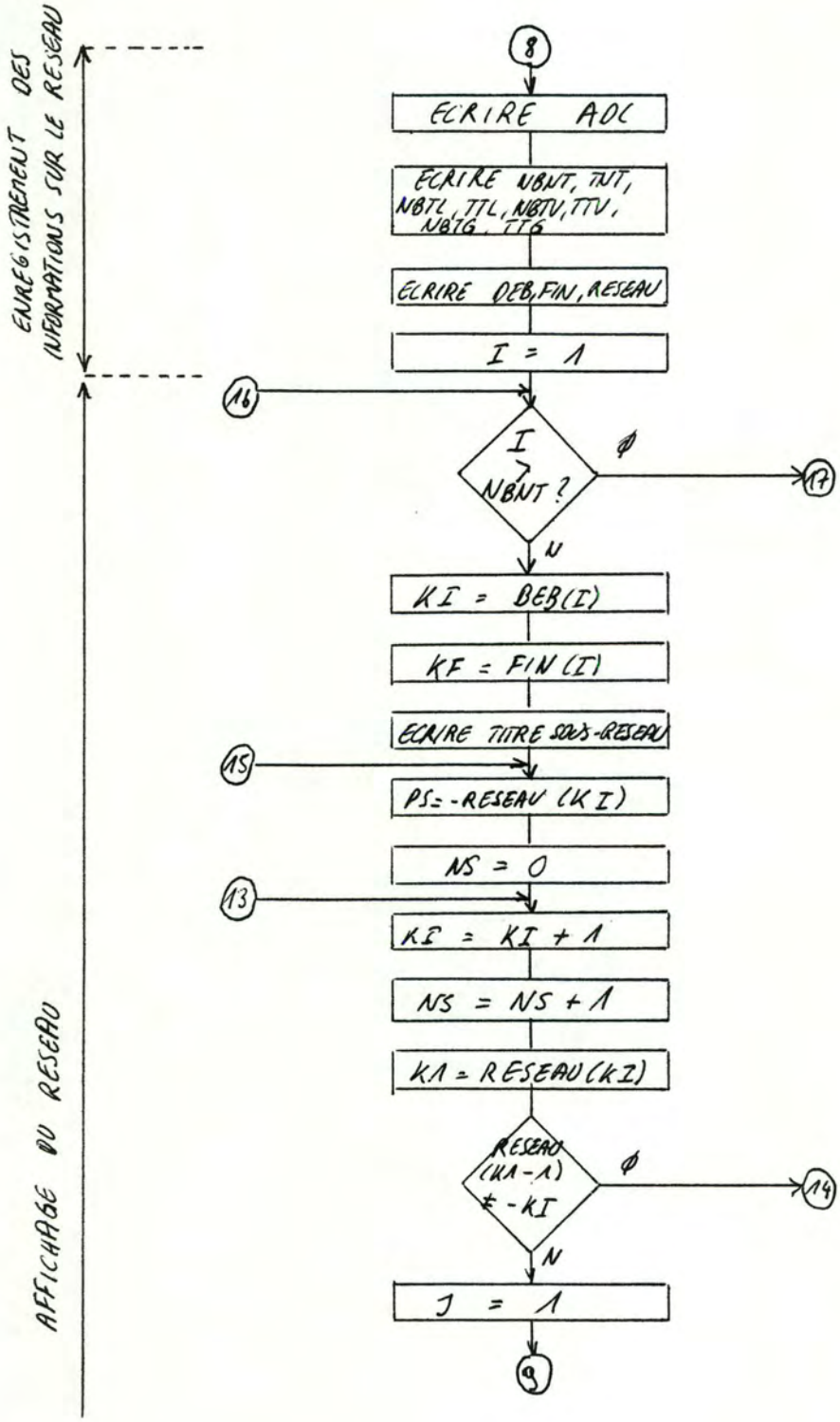


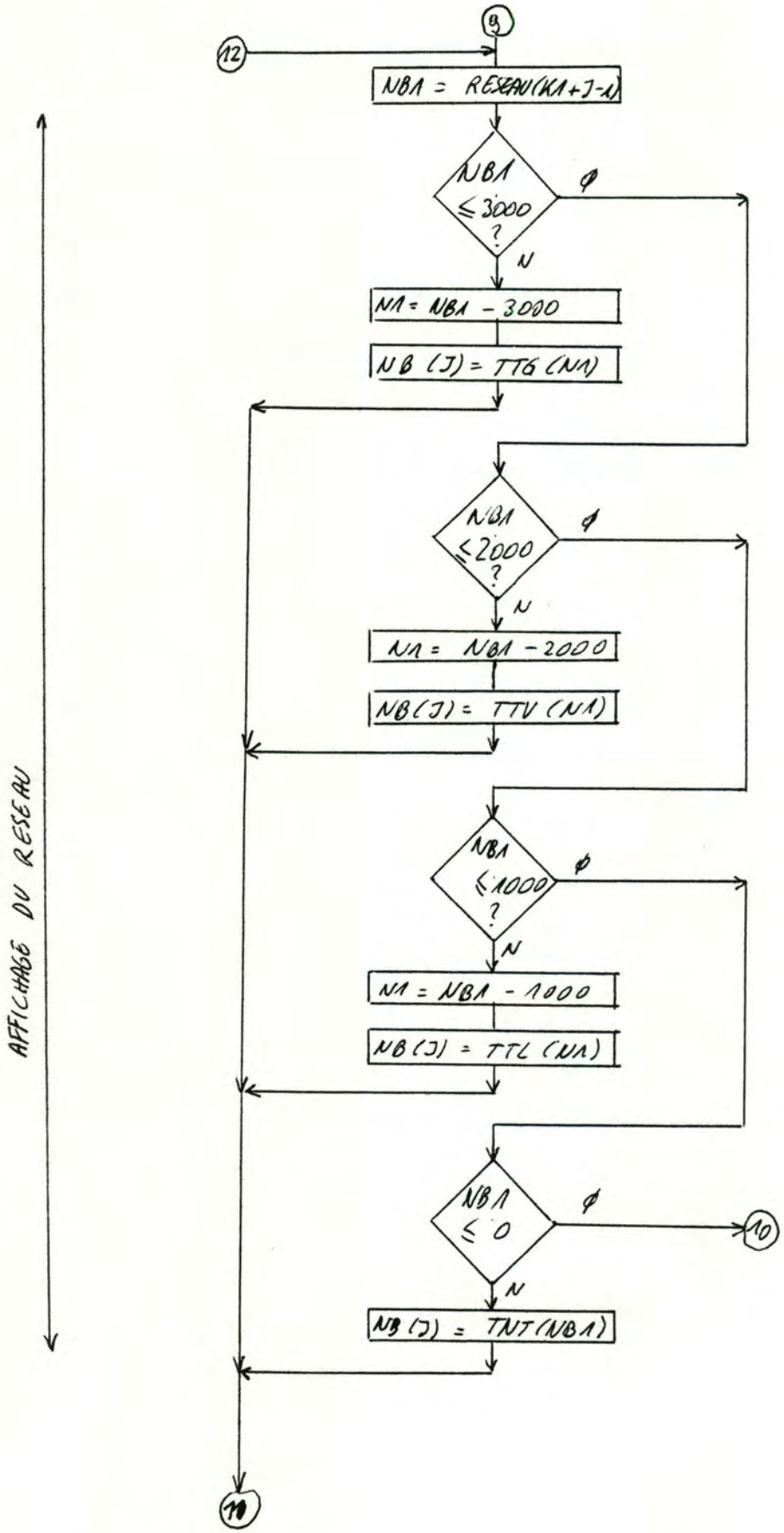

INITIALISATION D'UN SOUS-RESEAU
TRAITEMENT DES NOEUDS PROCEDURAUX
1
7
I>NBNT ?
Ø
8
N
LIRE NOM, PE, PS, NBP
ECRIRE NOM, PE, PS, NBP
J = 1
J >NBP ?
Ø
2
N
LIRE NUM, NBE, NBS
ECRIRE NUM, NBE, NBS
TAB (J,1) = NUM
TAB (J,2) =ADC+NBE
TAB (J,3) = NBE
TAB (J,4) = NBS
RESEAU (ADC+NBE) = -NUM
NNNN = NUM/10
NUM >0 ?
Ø
RESEAU (ADC+NBE) = -(NUM-NNNN * 10)
N
ADC = ADC + NBE + NBS + 1
J = J + 1

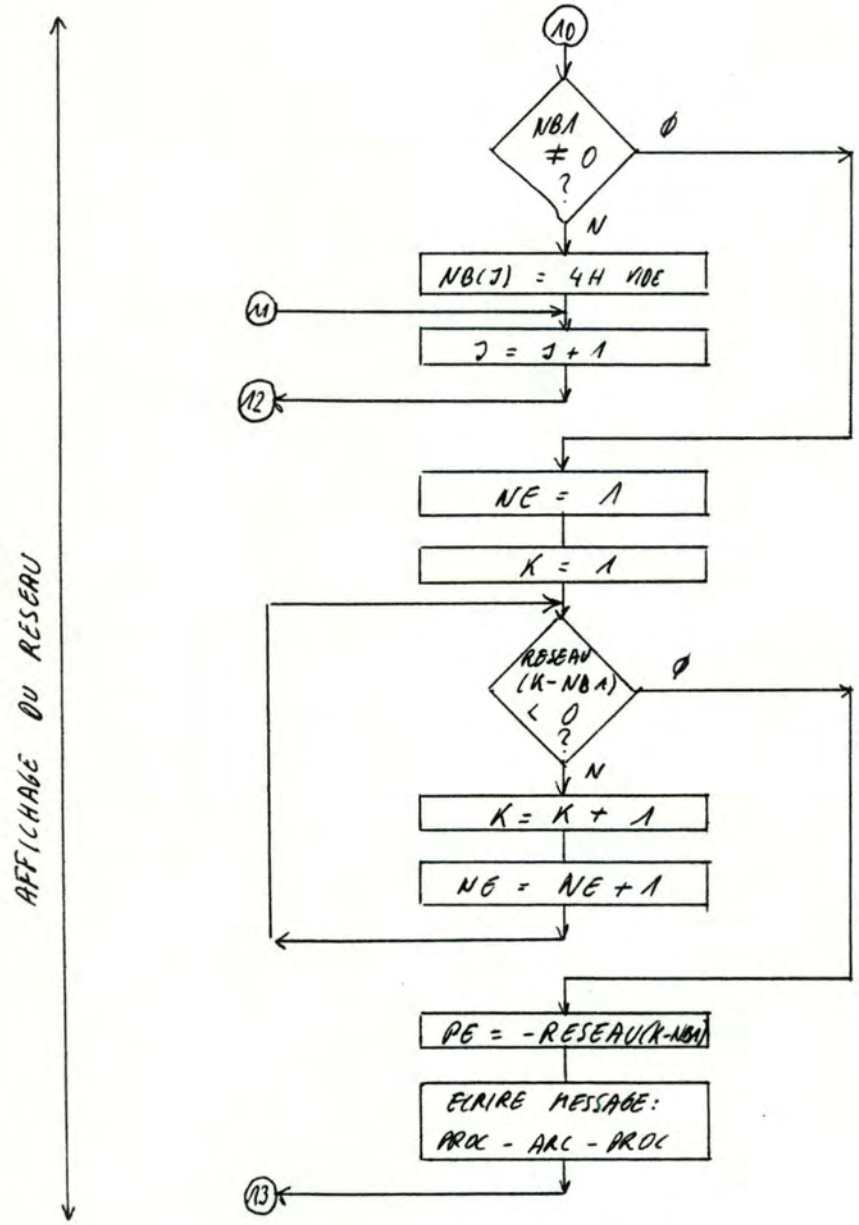

POSITIONNEMENT POINTEURS DEBUT-FIN DE CHAQUE RESEAU
TRAITEMENT DES BRANCHES LINEAIRES
2
APPEL PROCEDURE RECH (NOM, IND)
APPEL FONCTION AD DEB(IND) = AD(PE)
APPEL FONCTION AD FIN(IND) = AD(PS)
J = 1
5
J > NBP ?
Ø
6
N
NP = TAB(J,1)
ADR = TAB(J,2)
NBE = TAB(J,3)
NBS = TAB(J,4)
NBS = 0 ?
Ø
4
N
JK = 1
3

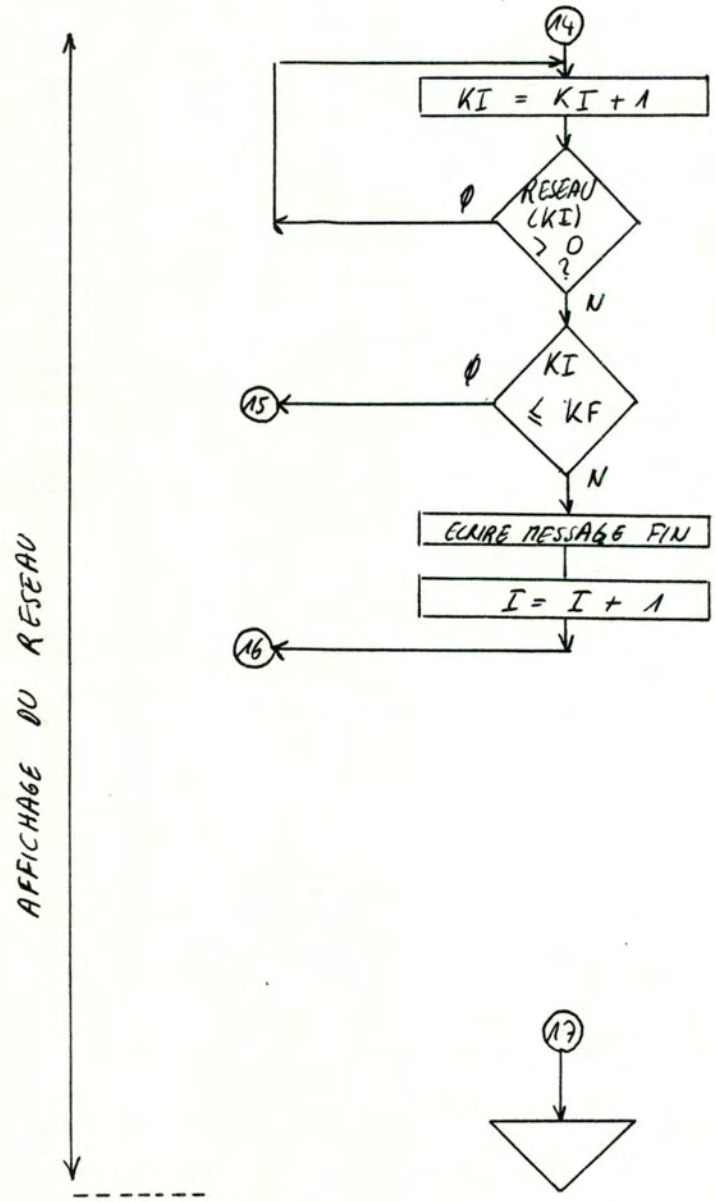

TRAITEMENT DES BRANCHES LINEAIRES
③
JK >NBS ?
Ø
④
N
LIRE PA, SA, E2, P2, NR(IA)
ECRIRE PA, SA, E2, P2, NR
RESEAU (ADR+ JK)=ADC+1
RESEAU (ADC)=-ADR-JK
ADC = ADC +1
KA = 1
NRA = NR (KA)
APPEL PROCEDURE RECH (NRA, IN)
RESEAU (ADC) = IN
ADC = ADC +1
KA = KA + 1
NRA = NR (KA)
Ø
NRA ≠ 4H
N
ADA = AD (P2)
RESEAU (ADC) = -ADA+E2
ADC = ADC + 1
RESEAU (ADA-E2)=ADC-2
JK = JK+1


TRAITEMENT DES
BRANCHES LINEAIRES
4
J = J + 1
5
6
I = I + 1
7


ENREGISTREMENT DES INFORMATIONS SUR LE RESEAU
AFFICHAGE DU RESEAU
8
ECRIRE ADC
ECRIRE NBNT, TNT, NBTL, TTL, NBTV, TTV, NBTG, TTG
ECRIRE DEB, FIN, RESEAU
I = 1
16
I > NBNT ?
Φ
17
N
KI = DEB(I)
KF = FIN(I)
ECRIRE TITRE SOUS-RESEAU
15
PS = -RESEAU (KI)
NS = 0
13
KI = KI + 1
NS = NS + 1
KA = RESEAU(KI)
RESEAU (KA-1) ≠ -KI
Φ
14
N
J = 1
9

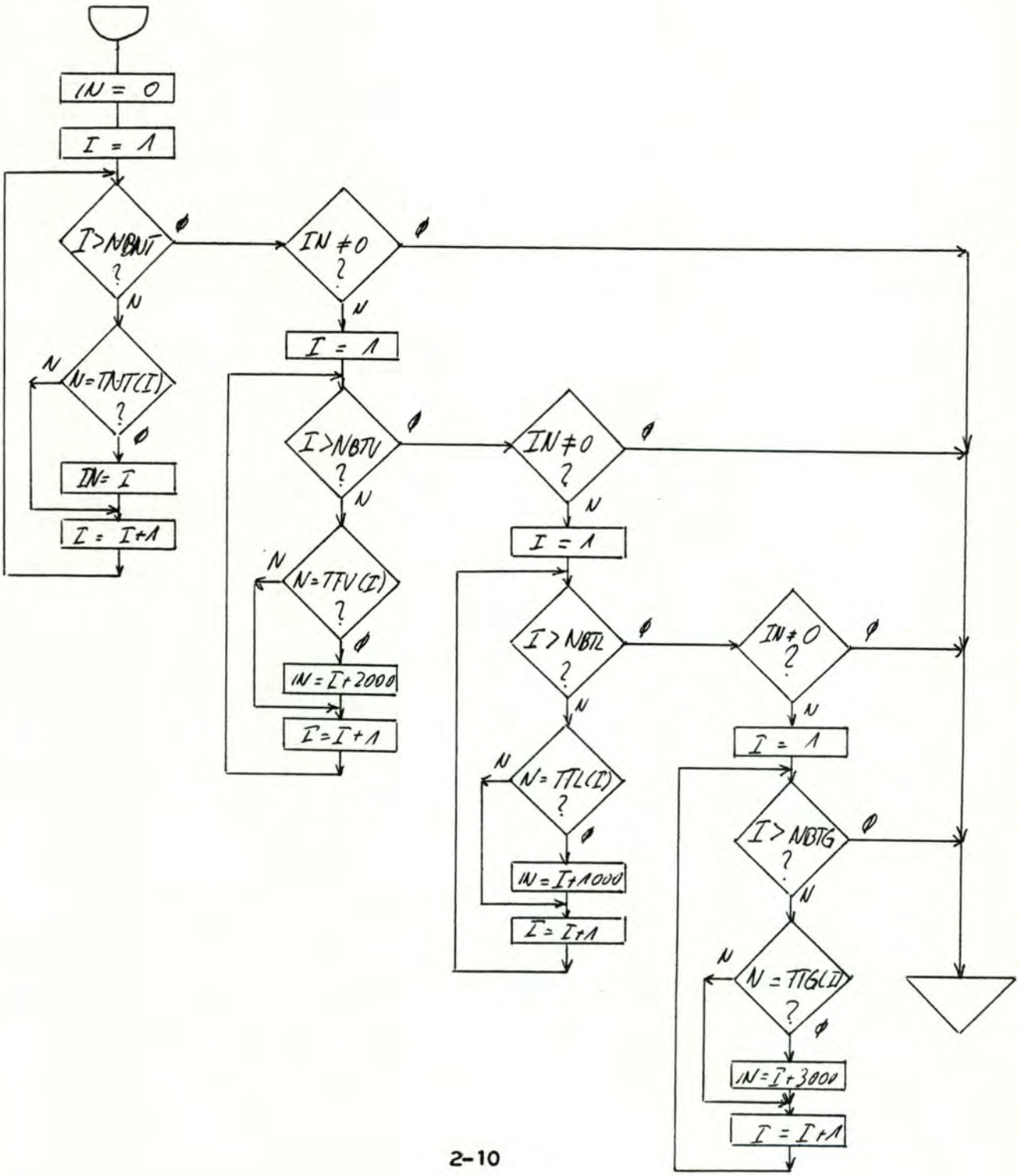

AFFICHAGE DU RESEAU
9
12
NB1 = RESEAU(K1+J-1)
NB1 ≤ 3000 ?
Φ
N
N1 = NB1 - 3000
NB (J) = TTG (N1)
NB1 ≤ 2000 ?
Φ
N
N1 = NB1 - 2000
NB(J) = TTV (N1)
NB1 ≤ 1000 ?
Φ
N
N1 = NB1 - 1000
NB(J) = TTL (N1)
NB1 ≤ 0
Φ
10
N
NB (J) = TNT(NB1)
10


AFFICHAGE DU RESEAU
10
NBA ≠ 0 ?
Ø
N
NB(J) = 4H VIDE
11
J = J + 1
12
NE = 1
K = 1
RESEAU (K-NBA) < 0 ?
Ø
N
K = K + 1
NE = NE + 1
PE = -RESEAU(K-NBA)
ECRIRE MESSAGE: PROC - ARC - PROC
13


AFFICHAGE DU RESEAU
14
KI = KI + 1
RESEAU (KI) > 0 ?
Ø
N
KI ≤ KF
Ø
15
N
ECRIRE MESSAGE FIN
I = I + 1
16
17

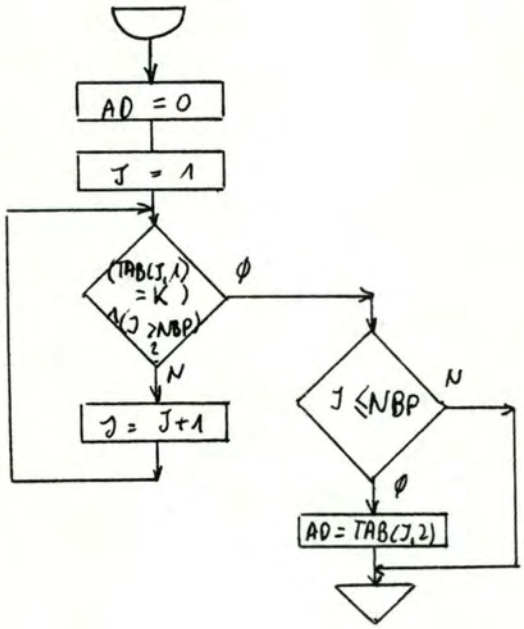
Procédure RECH(N,IN)

La fonction RECH recherche un élément correspondant à N dans les tableaux TNT, TTV, TTL, TTG. IN donne l'indice de l'élément et indique dans quel tableau il se trouve. (Si l'élément n'a pas été retrouvé, IN = 0).



Procédure AD(K)

La fonction AD recherche dans le tableau TAB l'élément correspondant au paramètre K. AD indique l'élément de la deuxième ligne de TAB correspondant à K.



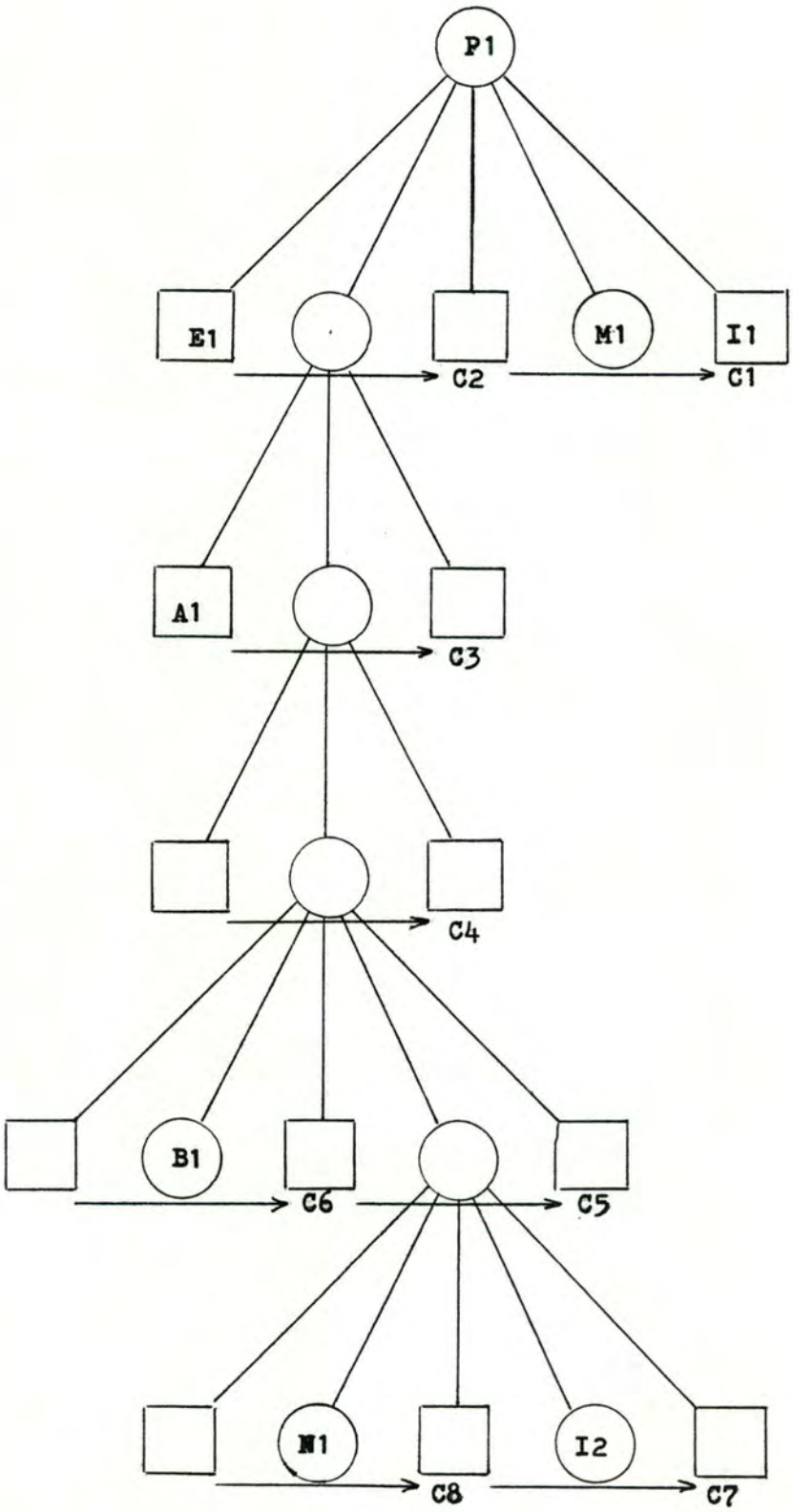
Comme nous l'avons déjà remarqué, nous conserverons de cet algorithme la structure fondamentale, càd la décomposition dans les étapes suivantes :

- Initialisation : Acquisition des valeurs de base du réseau comme :
  + nombre de non-terminaux
  + ensemble des non-terminaux
  + nombre de terminaux vrais
  + ensemble des terminaux vrais
  + nombre de terminaux lexique figés
  + ensemble des terminaux lexique figés
  + nombre de terminaux généraux
  + ensemble des terminaux généraux

  (nous y ajouterons une vérification visuelle par l'utilisateur ainsi qu'une possibilité de retour en arrière avec modification partielle)
- Traitement des sous-réseaux un par un : Ce traitement se décompose lui-même en trois sous-étapes :
  + Initialisation d'un sous-réseau, càd acquisition des valeurs de base d'un sous-réseau comme :
    - procédure d'entrée
    - procédure de sortie
    - nombre de procédures
  + Traitement des noeuds procéduraux de ce sous-réseau, où on définit pour chaque noeud procédural les caractéristiques suivantes :
    - nombre d'entrées
    - nombre de sorties
  + Traitement des branches linéaires de ce sous-réseau, où on définit pour chaque noeud procédural et pour chaque branche sortant de ce noeud :
    - la procédure d'arrivée de cette branche
    - le numéro de l'entrée dans cette procédure
    - les éléments composant cette branche

  (nous y ajouterons des vérifications automatiques sur la cohérence des données introduites ainsi que des options de retour en cas d'erreurs)
- Affichage du réseau et enregistrement sur fichier en vue d'une utilisation par un autre programme.
  (Ici aussi nous ajouterons une possibilité de retour en arrière)

Nous obtenons donc la structure suivante pour notre algorithme :



avec les significations suivantes :

- P1 : Programme principal
- I1 : Initialisation des valeurs de base du réseau
- M1 : Modification des valeurs de base du réseau
- I2 : Initialisation des valeurs de base d'un sous-réseau
- N1 : Traitement des noeuds procéduraux d'un sous-réseau
- B1 : Traitement des branches linéaires d'un sous-réseau
- A1 : Affichage du réseau
- E1 : Enregistrement du réseau sur un fichier
- C1 : Fin de modification des valeurs de base du réseau
- C2 : Description du réseau correcte
- C3 : Tous les sous-réseaux sont décrits
- C4 : Description du sous-réseau correcte
- C5 : Description noeuds et initialisation correctes
- C6 : Toutes les branches sont décrites ou reprise du sous-réseau
- C7 : Initialisation correcte
- C8 : Tous les noeuds sont décrits ou reprise du sous-réseau

Nous avons maintenant décrit la structure fondamentale de l'algorithme, la partie suivante donnera les détails du traitement.

2.2. DESCRIPTION DETAILLEE DES TRAITEMENTS

Dans les paragraphes qui vont suivre, nous allons donner la description détaillée de ce que devront réaliser les différentes parties dégagées dans la structure fondamentale.

2.2.1. Initialisation des valeurs de base du réseau

Dans cette partie, il s'agit d'initialiser les valeurs de base du réseau, càd :

- demander à l'utilisateur d'introduire les nombres de non-terminaux, terminaux vrais, - de type lexiques figés et - généraux ainsi que l'ensemble des non-terminaux, terminaux vrais, - de type lexiques figés et - généraux;
- saisir la réponse de l'utilisateur au terminal;
- vérifier la cohérence des données introduites par l'utilisateur;
- dans le cas d'erreurs, les signaler à l'utilisateur par des messages d'erreurs et gérer la reprise des traitements;
- dans le cas d'une demande de renseignements par l'utilisateur, lui répondre par un message 'help';
- donner à l'utilisateur la possibilité de vérifier visuellement les données introduites.

Cette initialisation n'est effectuée qu'une seule fois.

2.2.2. Modification des valeurs de base du réseau

Après l'initialisation des valeurs de base, il faut donner la possibilité à l'utilisateur de modifier ces valeurs, une erreur ayant été découverte lors de la vérification visuelle. Il faut donc :

- demander à l'utilisateur s'il veut modifier quelque chose;
- si oui, lui demander séparément s'il veut modifier les non-terminaux, terminaux vrais, - de type lexiques figés ou - généraux;
- pour chaque classe à modifier redemander le nombre et l'ensemble des éléments de cette classe;

- saisir la réponse de l'utilisateur au terminal;
- vérifier la cohérence des données introduites par l'utilisateur;
- dans le cas d'erreurs, les signaler à l'utilisateur par des messages d'erreurs et gérer la reprise des traitements;
- dans le cas d'une demande de renseignements par l'utilisateur, lui répondre par un message 'help';
- donner à l'utilisateur la possibilité de vérifier visuellement les données introduites.

Cette modification des valeurs de base du réseau doit pouvoir être faite autant de fois que le désire l'utilisateur. Ayant cependant décidé que cette modification était terminé, on n'y reviendra plus.

2.2.3. Initialisation des valeurs de base d'un sous-réseau

Ce module initialisera les valeurs de base d'un sous-réseau, càd :

- il devra demander à l'utilisateur d'introduire le nom du sous-réseau à traiter;
- saisir la réponse de l'utilisateur au terminal;
- vérifier si ce sous-réseau n'a pas encore été décrit :
  - s'il a déjà été décrit, le signaler à l'utilisateur et demander la réintroduction jusqu'à en obtenir un qui n'ait pas encore été décrit,
  - sinon accepter ce nom;
- demander à l'utilisateur le numéro de la procédure d'entrée, - de sortie et le nombre de procédures du sous-réseau;
- saisir la réponse de l'utilisateur au terminal;
- vérifier la cohérence de ces données (ex.: nbe ≠ nbs);
- s'il y a des erreurs, les signaler à l'utilisateur et demander la réintroduction des données erronnées;
- donner à l'utilisateur la possibilité de demander des renseignments et lui répondre par des messages 'help'.

A la fin de cette initialisation, il faut donner à l'utilisateur la possibilité de revenir en arrière et de modifier les données introduites. On ne passera à la partie suivante que si l'utilisateur en a décidé ainsi.

2.2.4. Traitement des noeuds procéduraux d'un sous-réseau

On traitera dans cette partie les informations relatives aux noeuds procéduraux du sous-réseau dont on a initialisé les valeurs de base dans la partie 'Initialisation des valeurs de base d'un sous-réseau'. Il faudra :

- demander à l'utilisateur d'introduire le numéro de la procédure qu'il désire décrire;
- demander à l'utilisateur de donner le nombre d'entrées et le nombre de sorties pour cette procédure;
- saisir les réponses de l'utilisateur au terminal;
- vérifier la cohérence des données introduites, notamment :
  + la première procédure à décrire doit être la procédure d'entrée,
  + la dernière procédure à décrire doit être la procédure de sortie,
  + aucune procédure ne peut être décrite deux fois,
  + la procédure d'entrée doit avoir 0 entrées, inversément, une procédure n'étant pas procédure d'entrée, ne peut avoir 0 entrées,
  + la procédure de sortie doit avoir 0 sorties, inversément, une procédure n'étant pas procédure de sortie, ne peut avoir 0 entrées,
  + toute procédure doit être décrite au moins une fois;
- dans le cas d'une erreur, la signaler à l'utilisateur par un message d'erreur;
- donner à l'utilisateur la possibilité de demander des renseignments et lui répondre par des messages 'help'.

Après le traitement de ces informations, il faudra vérifier leur cohérence à un niveau plus global (autant d'entrées que de sorties dans un même sous-réseau) et, dans le cas d'une erreur, permettre la reprise des traitements soit au niveau de l'initialisation des valeurs de base du sous-réseau, soit au niveau du traitement des noeuds procéduraux. Dans le cas où les données sont cohérentes, il faut quand même donner à l'utilisateur la possibilité de revenir en arrière pour modifier éventuellement des données. Pour cela, il doit pouvoir reprendre soit à l'initialisation des valeurs de base du sous-réseau, soit au traitement des noeuds procéduraux. Ce n'est que lorsque l'utilisateur le décide qu'on passe à la partie traitement des branches linéaires.

2.2.5. Traitement des branches linéaires d'un sous-réseau

Cette partie traitera les informations relatives aux branches linéaires du sous-réseau dont on a initialisé les valeurs de base dans la partie 'Initialisation des valeurs de base d'un sous-réseau' et dont on a décrit les noeuds procéduraux dans la partie 'Traitement des noeuds procéduraux d'un sous-réseau'. Il faudra :

- demander à l'utilisateur, pour une procédure donnée et une sortie de cette procédure également donnée, d'introduire le numéro de la procédure et le numéro de l'entrée dans cette procédure y correspondants;
- demander à l'utilisateur de décrire les éléments des branches ainsi définies;
- saisir les réponses de l'utilisateur au terminal;
- vérifier la cohérence des données introduites, notamment :
  + la procédure d'arrivée doit avoir été décrite dans la partie 'Traitement des noeuds procéduraux d'un sous-réseau';
  + la procédure d'arrivée ne peut être la procédure d'entrée dans le sous-réseau;
  + les entrées d'une procédure d'arrivée ne peuvent être décrites qu'une seule fois;
  + pour une procédure d'arrivée, on ne peut décrire plus d'entrées qu'on en a déclarées dans la partie 'Traitement des noeuds procéduraux d'un sous-réseau';
  + les éléments composant la branche linéaire doivent tous avoir été décrit lors de l'initialisation des valeurs de base du réseau;
- dans le cas d'une erreur, la signaler à l'utilisateur par un message d'erreur;
- donner à l'utilisateur la possibilité de demander des renseignements et lui répondre par des messages 'help'.

Après le traitement de ces informations, il faudra vérifier leur cohérence à un niveau plus global (toutes les entrées de toutes les procédures doivent avoir été décrites) et, dans le cas d'une erreur, permettre la reprise des traitements soit au niveau de l'initialisation des valeurs de base du sous-réseau, soit au niveau du traitement des noeuds procéduraux du sous-réseau, soit au niveau du traitement des branches linéaires du sous-réseau. Dans le cas où les données sont cohérentes, il faut quand même donner à l'utilisateur la possibilité de revenir en arrière afin de modifier éventuellement des données. Pour cela, il doit pouvoir reprendre soit à l'initialisation des valeurs de base du sous-réseau, soit à la partie traitement des noeuds procéduraux du sous-réseau, soit à la partie traitement des branches linéaires. Ce n'est que lorsque l'utilisateur le décide, qu'on passe à la partie sui-

vante, qui est l'initialisation des valeurs de base du sous-réseau suivant ou, si on vient de décrire le dernier sous-réseau, l'affichage du réseau au terminal en vue d'une vérification visuelle par l'utilisateur.

2.2.6. Affichage du réseau

Après avoir receuilli tous les renseignements sur le réseau, il faut afficher le réseau à l'écran pour permettre une vérification visuelle par l'utilisateur. Ce sera alors à l'utilisateur de décider s'il faut encore modifier des données ou non. S'il faut reprendre la description, on va alors la reprendre au niveau de l'initialisation des valeurs de base du premier sous-réseau et on va oublier tous les traitements effectués jusqu'alors. Sinon on passe à l'enregistrement du réseau sur fichier.

2.2.7. Enregistrement du réseau sur un fichier

Ce module ne fait rien d'autre que mémoriser tous les renseignements nécessaires à une bonne interprétation de la représentation interne du réseau, ainsi que la représentation interne elle même sur fichier. Le réseau ainsi mémorisé peut être exploité par un autre programme (ex.: Analyseur syntaxique).

3. IMPLEMENTATION

L'ensemble de la programmation a été réalisé en PASCAL standard. La structure de base est donc une structure de tableau. Nous retrouverons ci-dessous la description du code du programme CRERES dans sa dernière version du 20/10/82.

Une remarque préliminaire s'impose encore : Le PASCAL tel qu'il est implémenté sur l'Exormacs 68000 n'admet pas le passage du nom d'une procédure comme paramètre à une autre procédure. Ceci a quelque peu alourdi le code.

3.1. Le programme principal

Le programme principal reprend en gros la structure fondamentale telle qu'elle a été décrite au chapitre précédent. Il y a trois grandes parties :

- une partie Initialisation;
- une partie Traitement du réseau;
- une partie Mémorisation du réseau.

3.1.1. Initialisation

Dans cette partie s'effectue l'initialisation des valeurs de base du réseau. Pour cela le programme utilise quatre procédures pour initialiser séparément les non-terminaux, terminaux généraux, vrais ou à lexiques figés. Ce sont les procédures INITNT, INITTV, INITTL, INITTG. Pour faire des modifications éventuelles, une procédure MODINI est appelée et cela jusqu'à ce que l'utilisateur décide de continuer en répondant non à la demande de modification. Ensuite le programme effectue encore quelques initialisations lui-même avant de passer à la partie suivante.

3.1.2. Traitement du réseau

Après l'initialisation des valeurs de base du réseau, on peut traiter le réseau, sous-réseau par sous-réseau. C'est cette structure que reflète la partie Traitement du réseau. Le réseau est traité jusqu'à ce qu'il ait été validé par l'utilisateur. De plus, le réseau est traité jusqu'à ce que tous les sous-réseaux aient été traités. Un sous-réseau est traité jusqu'à ce qu'il ait été décrit correctement et validé par l'utilisateur. Le traitement d'un sous-réseau se décompose lui-même en Initialisation des valeurs de base du sous-réseau, Traitement des noeuds procéduraux du sous-réseau et Traitement des branches linéaires du sous-réseau. Comme on veut permettre des retours en arrière, il y a des validations après chaque étape du traitement. On ne passe à la partie Traitement des branches que lorsque l'Initialisation et le Traitement des moeuds a été validé. De même, on ne passe au Traitement des noeuds que lorsque l'Initialisation a été validée. Pour initialiser les valeurs de base, une procédure TRDSR est appelée. Le Traitement des noeuds procéduraux s'effectue par l'appel de la procédure TRNO. La cohérence des données introduites est vérifiée par la procédure VERNO. Après validation, on passe au Traitement des branches linéaires qui est réalisé par l'appel de la procédure TRBR, à la suite de laquelle la procédure de vérification VERBR est appelée pour vérifier la cohérence des données acquises. Tous les sous-réseaux ayant été traités, on affiche alors le réseau par la procédure IMPRE en vue d'une vérification visuelle par l'utilisateur. La validation s'effectue par la procédure VERRE dont le résultat décide de la suite du traitement, càd :

- passage à la partie suivante;
- ou reprise au début du traitement du réseau en oubliant tout ce qui a été décrit jusqu'alors.

3.1.3. Mémorisation du réseau

Cette partie se résume à l'appel de la procédure ECRRE qui écrit le réseau avec les informations annexes sur un fichier qui s'appelle RNP.

3.2. Les procédures d'initialisation

Nous appellerons procédures d'initialisation les procédures suivantes : INITNT, INITTV, INITTL, INITTG et MODINI.

3.2.1. INITNT

Cette procédure initialise le nombre de non-terminaux ainsi que l'ensemble des non-terminaux en faisant appel à la procédure LECTURE chargée de saisir les données au terminal. Elle fait le joint entre les résultats de LECTURE et la représentation en mémoire centrale de ces données utilisées tout au long du programme.

3.2.2. INITTV

Même rôle que INITNT, mais pour les terminaux vrais.

3.2.3. INITTL

Même rôle que INITNT, mais pour les terminaux à lexiques figés.

3.2.4. INITTG

Même rôle que INITNT, mais pour les terminaux généraux.

3.2.5. MODINI

Cette procédure donne à l'utilisateur la possibilité de modifier les données initialisées précédemment en lui demandant pour chaque catégorie de mots s'il désire une modification. Pour la réintroduction des données, la procédure utilise les procédures INITTV, INITTL, INITTG, INITNT.

3.3. Les procédures de traitement des sous-réseaux

3.3.1. TRDSR

Cette procédure sert à initialiser les valeurs de base d'un sous-réseau. Elle demande à l'utilisateur le nom du sous-réseau qu'il désire décrire. Ensuite elle vérifie qu'il s'agit bien d'un non-terminal et que ce non-terminal n'a pas encore été décrit. Après avoir demandé à l'utilisateur d'introduire le numéro de la procédure d'entrée, celui de la procédure de sortie et le nombre de noeuds procéduraux du sous-réseau, elle lui demande s'il veut modifier les données qu'il vient d'introduire. En fonction de la réponse de l'utilisateur, une variable est positionnée de manière à signaler au programme principal la décision de l'utilisateur.

3.3.2. TRNO

Dans cette procédure on va traiter les noeuds procéduraux du sous-réseau que l'utilisateur a décidé de décrire dans la procédure TRDSR. Il faudra décrire tous les noeuds procéduraux de ce sous-réseau en commençant par la procédure d'entrée dans le sous-réseau et en terminant par la procédure de sortie du sous-réseau. La procédure demande à l'utilisateur d'introduire le numéro de la procédure qu'il désire décrire. Elle vérifie que la description se fait au bon endroit et que le noeud n'a pas encore été décrit. Ensuite elle demande l'introduction du nombre d'entrées et du nombre de sorties du noeud procédural. La cohérence des données est vérifiée. S'il n'y a pas eu d'erreurs, on porte les informations receuillies dans la représentation interne du réseau, sinon on fait appel à la procédure REPNO qui remet toutes les variables dans un état déterminé qui dépend du niveau de reprise désiré par l'utilisateur.

3.3.3. TRBR

On traite ici les informations relatives au branches linéaires du sous-réseau que l'utilisateur a décidé de décrire dans la procédure TRDSR. La procédure demande de donner pour une procédure déterminée et une sortie de cette procédure, le numéro de la procédure avec le numéro de l'entrée y correspondants et les éléments constituant la branche ainsi déterminée. Elle vérifie que la procédure d'arrivée existe, que l'entrée dans cette procédure existe, que la procédure d'arrivée n'est

pas la procédure d'entrée dans le sous-réseau, que l'entrée n'ait pas encore été décrite... Pour l'acquisition et la vérification des éléments constituants la branche elle fera appel à la procédure INARC. S'il n'y a pas eu d'erreurs, on porte les informations receuillies dans la représentation interne du réseau, sinon on fait appel à la procédure REPBR qui remet toutes les variables dans un état déterminé qui dépend du niveau de reprise désiré par l'utilisateur.

3.4. Les procédures de vérification de cohérence

3.4.1. VERNO

Cette procédure vérifie la cohérence des données se rapportant aux noeuds procéduraux, à un niveau plus global. Elle vérifie que le nombre total d'entrées dans un sous-réseau est égal au nombre de sorties dans ce sous-réseau. De plus, si toutes les données sont cohérentes, elle donne quand même à l'utilisateur la possibilité de revenir en arrière pour modifier éventuellement des données. Si des reprises doivent être effectuées, elle fait appel à la procédure REPNO, sinon elle positionne une variable qui indique au programme principal la décision de l'utilisateur.

3.4.2. VERBR

Cette procédure vérifie la cohérence de la description des branches linéaires, en vérifiant que toutes les entrées ont été décrites. De plus, si toutes les données sont cohérentes, elle donne quand même à l'utilisateur la possibilité de revenir en arrière pour modifier éventuellement des données. Si des reprises doivent être effectuées, elle fait appel à la procédure REPBR, sinon elle positionne une variable qui indique au programme principal la décision de l'utilisateur.

3.4.3. VERRE

Ici on donne à l'utilisateur la possibilité de reprendre le traitement du réseau. S'il désire le reprendre, on oublie tout ce qui a été fait jusque maintenant, càd on réinitialise un certain nombre de variables de manière à ce qu'on puisse reprendre le traitement au moment de l'initialisation du premier sous-réseau.

3.5. Les procédures de gestion de reprise

3.5.1. REPNO

Cette procédure a comme objectif de remettre certaines variables dans un état déterminé par le niveau de reprise désiré par l'utilisateur. L'utilisateur a trois possibilités de reprises :

- au niveau Initialisation des valeurs de base du sous-réseau;
- au niveau Traitement des noeuds procéduraux du sous-réseau;
- au niveau de la description erronnée, si l'erreur a été détectée avant la vérification globale VERNO.

Elle doit donc demander à l'utilisateur laquelle de ces trois options lui convient et de modifier la représentation interne du réseau en conséquence.

3.5.2. REPBR

Cette procédure a comme objectif de remettre certaines variables dans un état déterminé par le niveau de reprise désiré par l'utilisateur. L'utilisateur a quatre possibilités de reprises :

- au niveau Initialisation des valeurs de base du sous-réseau;
- au niveau Traitement des noeuds procéduraux du sous-réseau;
- au niveau Traitement des branches linéaires du sous-réseau;
- au niveau de la description erronnée, si l'erreur a été détectée avant la vérification globale VERBR.

Elle doit donc demander à l'utilisateur laquelle de ces quatre possibilités lui convient et de modifier la représentation interne du réseau en conséquence.

3.6. Les procédures Help et d'affichage de messages

Ces procédures ne font rien d'autre que d'afficher des messages au terminal. Elles sont entièrement définies par le contenu de ces messages et ne seront donc pas détaillées.

3.7. Les procédures utilitaires

3.7.1. ININT

Le rôle de cette procédure est de lire un entier au terminal et de le fournir à la procédure appelante ainsi qu'un code retour indiquant si l'utilisateur n'a rien tapé, s'il désire des précisions, s'il a tapé qch d'insignifiant ou si la lecture a réussi.

3.7.2. INOUI

La procédure lit une réponse au terminal qui doit soit être oui, soit non. Elle doit fournir la réponse à la procédure appelante ainsi qu'un code retour indiquant si l'utilisateur n'a rien tapé, s'il désire des précisions, s'il a tapé qch d'insignifiant ou si la lecture a réussi.

3.7.3. LECTURE

La procédure LECTURE lit un nombre et une liste d'éléments au terminal. Le nombre lu détermine le nombre d'éléments de la liste. En cas d'incohérence entre les deux, l'utilisateur peut réintroduire la liste soit réintroduire le nombre et la liste. S'il n'y plus d'incohérence, la liste est envoyée à la procédure appelante de même que le nombre.

3.7.4. RECH

La procédure RECH doit rechercher un élément dans les tableaux tnt, ttv, ttl et ttg. Elle doit indiquer à la procédure appelante si l'élément se trouve dans un de ces tableaux et, si oui, dans lequel et à quel endroit.

3.7.5. INSTRING

La procédure INSTRING doit lire une chaîne de caractères au terminal et fournir cette chaîne à la procédure appelante ainsi qu'un code retour indiquant si l'utilisateur n'a rien tapé, si l'utilisateur veut des précisions, si l'utilisateur a tapé qch d'insignifiant ou si la lecture a réussi.

3.7.6. INARC

La procédure INARC doit lire une liste d'éléments au terminal et fournir cette liste à la procédure appelante ainsi qu'un code retour indiquant si l'utilisateur n'a rien tapé, si l'utilisateur veut des précisions, si l'utilisateur a tapé qch d'insignifiant ou si la lecture a réussi.

3.8. Les procédures d'affichage et de mémorisation du réseau

3.8.1. IMPRE

Cette procédure doit parcourir la représentation interne du réseau à noeuds procéduraux et afficher ces renseignements sous une forme lisible au terminal de manière à permettre une vérification visuelle par l'utilisateur. L'affichage se fait sous-réseau par sous-réseau.

3.8.2. ECRRE

Cette procédure doit écrire la représentation interne du réseau à noeuds procéduraux accompagnée des informations annexes sur un fichier qui s'appelle RNP. Cette représentation interne pourra ainsi être exploitée par un autre programme.

4. LISTING SOURCE

REMARQUE

La différence entre la version du 20/10/82 et celle du 8/11/82 réside dans le fait que la première version admet des entrées pour une procédure d'entrée dans un sous-réseau et des sorties pour une procédure de sortie d'un sous-réseau, tandis que la deuxième version n'admet pas ces possibilités et signale une erreur lorsque le cas se présente. Cette version ne se distingue dès lors de la première que par deux tests supplémentaires venant s'ajouter aux deux tests de la page 36 (du listing) et qui deviennent :

```
  :
IF (((NBE = 0) AND (NUM ≠ PE)) OR
    ((NBE ≠ 0) AND (NUM = PE)))
THEN BEGIN
       :


       :
       IF (((NBS = 0) AND (NUM ≠ PS)) OR
           ((NBS ≠ 0) AND (NUM = PS)))
       THEN BEGIN
              :
```

A part cela rien n'est modifié dans le programme.

A N N E X E II

Listing du programme CRERES avec résultats

```
PAGE     1    LIST VERSION  120281 3  11/ 8/82  15:44:37   CRERES

(*************************************************************************)
(*                                                                       *)
(*                            PROGRAMME CRERES                           *)
(*                            ================                           *)
(*                                                                       *)
(*************************************************************************)
(*                                                                       *)
(* AUTEUR                        : MOUSEL Pierre                         *)
(* DATE DERNIERE MODIFICATION    : 08/11/82                              *)
(*                                                                       *)
(*************************************************************************)

program creres(input,output);

(*************************************************************************)
(*                                                                       *)
(*                 DECLARATION DES DONNEES GLOBALES                      *)
(*                 --------------------------------                      *)
(*                                                                       *)
(*************************************************************************)

const bo1nt       = 20;    (* nombre maximal de non-terminaux            *)
      bo2nt       = 4;     (* nombre maximal de caracteres/non-terminal  *)
      bo1tv       = 20;    (* nombre maximal de terminaux vrais          *)
      bo2tv       = 4;     (* nombre maximal de caracteres/terminal vrai *)
      bo1tl       = 30;    (* nombre maximal de terminaux lexiques figes *)
      bo2tl       = 4;     (* nombre maximal de caracteres/term.lex.fig. *)
      bo1tg       = 20;    (* nombre maximal de terminaux generaux       *)
      bo2tg       = 4;     (* nombre maximal de caracteres/terminal general *)
      bores       = 2000;  (* nombre maximal d'elements dans le reseau   *)
      nproc       = 20;    (* nombre maximal de noeuds par sous-reseau   *)
      boarc       = 10;    (* nombre maximal d'arcs par branche          *)

type  tableau     = array[1..120] of char;
      sauvproc    = array[1..nproc,1..5] of integer;
      codret      = (blank,help,invalid,ok);
      chaine      = array[1..bo2nt] of char;
      elnt        = array[1..bo2nt] of char;
      eltv        = array[1..bo2tv] of char;
      eltl        = array[1..bo2tl] of char;
      eltg        = array[1..bo2tg] of char;
      arcs        = array[1..bo2nt] of char;
      tablarc     = array[1..boarc] of arcs;

var   tab         : tableau; (* intermediaire ecran/tnt,ttv,ttl,ttg      *)
      nom         : chaine;  (* contient le nom du sous-reseau en        *)
                             (* cours de traitement                      *)
      code        : codret;  (* test validite reponse                    *)
      table       : sauvproc;(* tableau de sauvegarde                    *)
      arc         : arcs;    (* contiendra les noms des arcs             *)
      tarc        : tablarc; (* contient la liste des arcs d'une branche *)
      res         : text;    (* fichier qui contiendra les resultats     *)
      nbnt        : integer; (* nombre de non-terminaux lus              *)
      nbtv        : integer; (* nombre de terminaux vrais lus            *)
      nbtl        : integer; (* nombre de terminaux lexiques figes lus   *)
      nbtg        : integer; (* nombre de terminaux generaux lus         *)
      i,j,k,l,m   : integer; (* entiers servant d'index                  *)
```

```
      ind          : integer; (* indice d'un non-terminal dans tnt          *)
      pe           : integer; (* procedure d'entree pour un sous-reseau     *)
      ps           : integer; (* procedure de sortie pour un sous-reseau    *)
      nbp          : integer; (* nombre de procedures pour un sous-reseau   *)
      num          : integer; (* numero de la procedure                     *)
      nbe          : integer; (* nombre d'entrees de la procedure           *)
      nbs          : integer; (* nombre de sorties de la procedure          *)
      adc          : integer; (* premier element disponible dans reseau     *)
      nnnn         : integer; (* var. de trav. pour procedures dupliciees   *)
      adr          : integer; (* adresse d'une procedure dans le reseau     *)
      ent          : integer; (* numero de l'entree dans une procedure      *)
      proc         : integer; (* numero de la procedure ds laquelle on entre *)
      indproc      : integer; (* indice dans table de cette procedure       *)
      indarc       : integer; (* indice ds tnt, ttv, ttl ou ttg d'un arc    *)
      decrit       : array[1..bolnt] of boolean; (* non-terminaux traites   *)
      reseau       : array[1..bores] of integer; (* repres. int. du reseau  *)
      deb          : array[1..bolnt] of integer; (* ptrs vers entrees ds s-r *)
      fin          : array[1..bolnt] of integer; (* ptrs vers sorties de s-r *)
      tnt          : array[1..bolnt] of elnt;    (* liste des n-t            *)
      ttv          : array[1..boltv] of eltv;    (* liste des t-v            *)
      ttl          : array[1..boltl] of eltl;    (* liste des t-l            *)
      ttg          : array[1..boltg] of eltg;    (* liste des t-g            *)
      ch           : char;    (* intermediaire tnt,ttv,ttl,ttg/ecran         *)
      reponse      : char;    (* contient reponse si valide                  *)
      repinvalide  : boolean; (* controle validite reponse                   *)
      rincor       : boolean; (* controle reseau correct                     *)
      srincor      : boolean; (* controle sous-reseau correct                *)
      nincor       : boolean; (* controle noeud correct                      *)
      dsrincor     : boolean; (* controle description sous-reseau correcte   *)
      nomincor     : boolean; (* controle nom correct                        *)
      reprsr       : boolean; (* controle reprise sous-reseau                *)
      reprbr       : boolean; (* controle reprise description des branches   *)
      erreur       : boolean; (* controle description procedures correcte    *)
      dernier      : boolean; (* controle impression sous-reseau             *)

(*****************************************************************************)
(*                                                                           *)
(*                          DECLARATION DE PROCEDURES                        *)
(*                          -------------------------                        *)
(*                                                                           *)
(*****************************************************************************)

(*****************************************************************************)
(*                                                                           *)
(*              PROCEDURES HELP ET D'AFFICHAGE DE MESSAGES                   *)
(*                                                                           *)
(*****************************************************************************)

procedure vidoui;
begin
writeln;
writeln ('# VOUS N''AVEZ RIEN TAPE.');
writeln ('? REPONDEZ OUI OU NON :');
writeln
end;

procedure invoui;
```

```
begin
writeln;
writeln ('# VOTRE REPONSE EST INVALIDE.');
writeln ('? REPONDEZ OUI OU NON :');
writeln
end;

procedure vidint;
begin
writeln;
writeln ('# VOUS N''AVEZ RIEN TAPE.');
writeln ('? VEUILLEZ ENTRER UN ENTIER :');
writeln;
end;

procedure invint;
begin
writeln;
writeln ('# VOTRE REPONSE EST INVALIDE.');
writeln ('? VEUILLEZ ENTRER UN ENTIER :');
writeln
end;

procedure vidstring;
begin
writeln;
writeln ('# VOUS N''AVEZ RIEN TAPE.');
writeln ('? VEUILLEZ ENTRER DES CARACTERES SIGNIFICATIFS :');
writeln
end;

procedure invstring;
begin
writeln;
writeln ('# VOUS AVEZ INTRODUIT PLUSIEURS CHAINES DE CARACTERES.');
writeln ('? VEUILLEZ N''ENTRER QU''UNE SEULE CHAINE DE CARACTERES :');
writeln
end;

procedure mess0;
begin
writeln ('!-----------------------------------------------------------------------!');
writeln ('!                                                                       !');
writeln ('!                          PROGRAMME CRERES                             !');
writeln ('!                          ================                             !');
writeln ('!                                                                       !');
writeln ('! VOUS TRAVAILLEZ AVEC LE PROGRAMME CRERES, QUI EST UN LOGICIEL        !');
writeln ('! QUI VOUS PERMET DE PASSER INTERACTIVEMENT DE LA REPRESENTA-           !');
writeln ('! TION GRAPHIQUE D''UN RNP A SA REPRESENTATION INTERNE.                 !');
writeln ('!                                                                       !');
writeln ('! POUR PLUS DE RENSEIGNEMENTS, VOIR MANUEL D''UTILISATION.              !');
writeln ('!                                                                       !');
writeln ('!-----------------------------------------------------------------------!');
writeln ('!                                                                       !');
writeln ('!                      INITIALISATION DU RESEAU                         !');
writeln ('!                                                                       !');
writeln ('!-----------------------------------------------------------------------!')
```

```
PAGE     4    LIST VERSION  120281 3  11/ 8/82  15:44:37   CRERES

end;

procedure mess1;
begin
writeln;
writeln ('? VEUILLEZ ENTRER LE NOMBRE D''ELEMENTS :');
writeln
end;

procedure mess2(borne : integer);
begin
writeln;
writeln ('# VOTRE NOMBRE DEPASSE LA LIMITE DE :',borne:2,'.');
writeln ('? VEUILLEZ EN ENTRER UN AUTRE, PLUS PETIT :');
writeln
end;

procedure mess3;
begin
writeln;
writeln ('# VOTRE NOMBRE DOIT ETRE > 0.');
writeln ('? VEUILLEZ EN ENTRER UN AUTRE, PLUS GRAND :');
writeln
end;

procedure mess4;
begin
writeln;
writeln ('? VEUILLEZ ENTRER LES ELEMENTS, SEPARES PAR DES BLANCS, CHAQUE');
writeln ('? ELEMENT SE DISTINGUANT PAR LES QUATRE PREMIERS CARACTERES');
writeln ('? DES AUTRES ELEMENTS :');
writeln
end;

procedure mess5(borne : integer);
begin
writeln;
writeln ('# VOUS AVEZ DEPASSE LA LIMITE DE :',borne:2,' ELEMENTS.');
writeln
end;

procedure mess6;
begin
writeln;
writeln ('# VOUS N''AVEZ RIEN TAPE.');
writeln ('? VEUILLEZ ENTRER DES ELELMENTS :');
writeln
end;

procedure mess7;
begin
writeln;
writeln ('# VOTRE NOMBRE D''ELEMENTS INDIQUE, NE CORRESPOND PAS AU NOM-');
writeln ('# BRE D''ELEMENTS FOURNIS. NOUS ALLONS REPRENDRE.');
writeln ('? VOULEZ VOUS REPRENDRE AU NIVEAU DU NOMBRE : (O/N)');
writeln
end;
```

```
procedure mess8;
begin
writeln;
writeln ('? DONNEZ LA NOUVELLE LISTE DES ELEMENTS :');
writeln
end;

procedure mess9(borne : integer);
begin
writeln;
writeln ('# VOUS AVEZ DONNE TROP D''ARCS. UNE BRANCHE NE PEUT ETRE COM-');
writeln ('# POSEE QUE DE :',borne:2,' ARCS AU PLUS.');
writeln
end;

procedure mess10(arc : arcs);
var i  : integer;
begin
writeln;
write ('# L''ARC : ');
for i := 1 to bo2nt do write(arc[i]);
writeln (', N''A PAS ETE DEFINI DANS L''INITIALISATION DES');
writeln ('# NON-TERMINAUX, TERMINAUX VRAIS, GENERAUX OU FIGES.');
writeln
end;

procedure mess11;
begin
writeln;
writeln ('!        DEFINITION DES NON-TERMINAUX');
writeln ('!        ----------------------------');
writeln
end;

procedure mess12;
begin
writeln;
writeln ('!        DEFINITION DES TERMINAUX VRAIS');
writeln ('!        ------------------------------');
writeln
end;

procedure mess13;
begin
writeln;
writeln ('!        DEFINITION DES TERMINAUX LEXIQUES FIGES');
writeln ('!        --------------------------------------');
writeln
end;

procedure mess14;
begin
writeln;
writeln ('!        DEFINITION DES TERMINAUX GENERAUX');
writeln ('!        --------------------------------');
writeln
```

```
PAGE     6    LIST VERSION  120281 3  11/ 8/82  15:44:37   CRERES

end;

procedure mess15;
(* La procedure MESS15 affiche les non-terminaux,terminaux vrais, *)
(* terminaux lexiques figes et terminaux generaux dans un but de  *)
(* verification par l'utilisateur.                                *)
var i,j : integer;
begin
writeln;
writeln;
writeln;
writeln;
writeln ('!                    VERIFICATION');
writeln ('!                    ------------');
writeln;
writeln ('!                    NON-TERMINAUX :');
for i := 1 to bo1nt do
     begin
     for j := 1 to bo2nt do write(tnt[i,j]);
     write (' ')
     end;
writeln;
writeln ('!                    TERMINAUX VRAIS :');
for i := 1 to bo1tv do
     begin
     for j := 1 to bo2tv do write(ttv[i,j]);
     write (' ')
     end;
writeln;
writeln ('!                    TERMINAUX LEXIQUES FIGES :');
for i := 1 to bo1tl do
     begin
     for j := 1 to bo2tl do write(ttl[i,j]);
     write (' ')
     end;
writeln;
writeln('!                    TERMINAUX GENERAUX :');
for i := 1 to bo1tg do
     begin
     for j := 1 to bo2tg do write(ttg[i,j]);
     write (' ')
     end;
writeln
end;

procedure mess16;
begin
writeln;
writeln ('? VOULEZ VOUS MODIFIER QUELQUE CHOSE : (O/N)');
writeln
end;

procedure mess17;
begin
writeln;
writeln ('? VOULEZ VOUS MODIFIER LES NON-TERMINAUX : (O/N)');
writeln
```

```
end;

procedure mess18;
begin
writeln;
writeln ('? VOULEZ VOUS MODIFIER LES TEMINAUX VRAIS : (O/N)');
writeln
end;

procedure mess19;
begin
writeln;
writeln ('? VOULEZ VOUS MODIFIER LES TERMINAUX LEXIQUES FIGES : (O/N)');
writeln
end;

procedure mess20;
begin
writeln;
writeln ('? VOULEZ VOUS MODIFIER LES TERMINAUX GENERAUX : (O/N)');
writeln
end;

procedure mess21;
begin
writeln;
write ('? VOULEZ VOUS REPRENDRE AU NIVEAU INITIALISATION DU SOUS-RESEAU :');
writeln (' (O/N)');
writeln
end;

procedure mess22;
begin
writeln;
writeln ('? VOULEZ VOUS REPRENDRE  LA DESCRIPTION DES NOEUDS : (O/N)');
writeln
end;

procedure mess23;
begin
writeln;
writeln ('? VOULEZ VOUS REPRENDRE AVANT LE TRAITEMENT DES BRANCHES : (O/N)');
writeln
end;

procedure mess24;
begin
writeln;
writeln ('? VOULEZ VOUS REPRENDRE LA DESCRIPTION DES BRANCHES : (O/N)');
writeln
end;

procedure mess24c;
begin
writeln;
writeln;
writeln;
```

```
writeln;
writeln;
writeln ('!----------------------------------------------------------------------!');
writeln ('!                                                                      !');
writeln ('!                      TRAITEMENT DU RESEAU                            !');
writeln ('!                                                                      !');
writeln ('!----------------------------------------------------------------------!');
writeln
end;

procedure mess25 (i : integer);
begin
writeln;
writeln ('!            ------------------------------------');
writeln ('!            TRAITEMENT DU',i:3,'EME SOUS-RESEAU.');
writeln ('!            ------------------------------------');
writeln;
writeln;
writeln ('!             INITIALISATION DU SOUS-RESEAU.');
writeln ('!             ------------------------------');
writeln;
writeln;
writeln ('? NOM DU SOUS-RESEAU :');
writeln
end;

procedure mess26;
begin
writeln;
writeln ('# CE NOM NE FIGURE PAS PARMI L''ENSEMBLE DES NON-TERMINAUX.');
writeln ('? ENTREZ EN UN AUTRE :');
writeln
end;

procedure mess27;
begin
writeln;
writeln ('# CE SOUS-RESEAU A DEJA ETE DECRIT.');
writeln ('? DECRIVEZ EN UN AUTRE. ENTREZ LE NOM :');
writeln
end;

procedure mess28;
begin
writeln;
writeln ('? PROCEDURE D''ENTREE :');
writeln
end;

procedure mess29;
begin
writeln;
writeln ('? PROCEDURE DE SORTIE :');
writeln
end;

procedure mess29b;
```

```
PAGE     9    LIST VERSION  120281 3  11/ 8/82  15:44:37   CRERES

begin
writeln;
writeln ('# UNE PROCEDURE NE PEUT ETRE A LA FOIS PROCEDURE D''ENTREE ET PRO-');
writeln ('# CEDURE DE SORTIE D''UN MEME SOUS-RESEAU.');
writeln ('? INDIQUEZ LA PROCEDURE DE SORTIE :');
writeln
end;

procedure mess30;
begin
writeln;
writeln ('? NOMBRE DE NOEUDS-PROCEDURAUX :');
writeln
end;

procedure mess31;
begin
writeln;
writeln ('# IL FAUT AVOIR PLUS D''UN NOEUD DANS UN SOUS-RESEAU.');
writeln ('? DONNEZ LE NOMBRE DE NOEUDS :');
writeln
end;

procedure mess31b;
begin
writeln;
writeln ('# ON NE PEUT AVOIR PLUS DE ',nproc:2,' NOEUDS DANS UN SOUS-RESEAU.');
writeln ('? DONNEZ UN NOMBRE PLUS PETIT :');
writeln
end;

procedure mess32;
begin
writeln;
writeln ('? VOULEZ VOUS MODIFIER UNE DONNEE INTRODUITE: (O/N)');
writeln
end;

procedure mess32b;
begin
writeln;
writeln ('!            TRAITEMENT DES NOEUDS DU SOUS-RESEAU');
writeln ('!            ------------------------------------');
writeln
end;

procedure mess33(i : integer);
begin
writeln;
writeln ('?          ',i:2,'. PROCEDURE NUMERO :');
writeln
end;

procedure mess34;
begin
writeln;
writeln ('# DONNEE INVALIDE. CETTE PROCEDURE A DEJA ETE DECRITE.');
```

```
writeln
end;

procedure mess35;
begin
writeln;
writeln ('? NOMBRE D''ENTREES :');
writeln
end;

procedure mess36;
begin
writeln;
writeln ('# DONNEE INVALIDE. UNE PROCEDURE N''ETANT PAS UNE PROCEDURE');
writeln ('# D''ENTREE, DOIT AVOIR PLUS DE 0 ENTREES. INVERSEMENT, UNE');
writeln ('# PROCEDURE ETANT PROCEDURE D''ENTREE, NE PEUT AVOIR PLUS DE');
writeln ('# DE 0 ENTREES.');
writeln
end;

procedure mess37;
begin
writeln;
writeln ('? NOMBRE DE SORTIES :');
writeln
end;

procedure mess38;
begin
writeln;
writeln ('# DONNEE INVALIDE. UNE PROCEDURE N''ETANT PAS UNE PROCEDURE');
writeln ('# DE SORTIE, DOIT AVOIR PLUS DE 0 SORTIES. INVERSEMENT, UNE');
writeln ('# PROCEDURE ETANT UNE PROCEDURE DE SORTIE, NE PEUT AVOIR PLUS');
writeln ('# DE 0 SORTIES.');
writeln
end;

procedure mess38b;
begin
writeln;
writeln ('# VOUS N''AVEZ PAS ENCORE DECRIT LA PROCEDURE D''ENTREE.');
writeln
end;

procedure mess38c;
begin
writeln;
writeln ('# LA DERNIERE PROCEDURE DECRITE DOIT ETRE LA PROCEDURE DE SORTIE.');
writeln
end;

procedure mess38d;
begin
writeln;
writeln ('# LA PROCEDURE D''ENTREE DOIT ETRE DECRITE LA PREMIERE TANDIS QUE');
writeln ('# LA PROCEDURE DE SORTIE DOIT ETRE DECRITE LA DERNIERE.');
writeln
```

```
end;

procedure mess38e;
begin
writeln;
writeln ('!            TRAITEMENT DES BRANCHES LINEAIRES.');
writeln ('!            ---------------------------------');
writeln
end;

procedure mess39(i,num : integer);
begin
writeln;
writeln ('!          ',i:2,'. PROCEDURE : ',num:2);
writeln
end;

procedure mess40(k : integer);
begin
writeln;
writeln ('? SORTIE : ',k:2,' VERS ENTREE :');
writeln
end;

procedure mess41;
begin
writeln;
writeln ('? DE LA PROCEDURE :');
writeln
end;

procedure mess42;
begin
writeln;
writeln ('# CETTE PROCEDURE N''A PAS ETE DECRITE.');
writeln
end;

procedure mess43;
begin
writeln;
writeln ('# ON NE PEUT ENTRER DANS UNE PROCEDURE D''ENTREE.');
writeln
end;

procedure mess44(p,e : integer);
begin
writeln;
writeln ('# LA PROCEDURE ',p:2,' N''A PAS ',e:2,' ENTREES.');
writeln
end;

procedure mess45(p : integer);
begin
writeln;
writeln ('# LES ENTREES DE LA PROCEDURE ',p:2,' ONT TOUTES ETE DECRITES.');
writeln
```

```
end;

procedure mess46;
begin
writeln;
writeln ('# CETTE ENTREE A DEJA ETE DECRITE.');
writeln
end;

procedure mess46b;
begin
writeln;
writeln ('? EN PASSANT PAR LES ARCS :');
writeln
end;

procedure mess47;
begin
writeln;
writeln ('# VOTRE NOMBRE TOTAL D''ENTREES DE CE SOUS-RESEAU EST DIFFERENT');
writeln ('# DU NOMBRE TOTAL DE SORTIES DU SOUS-RESEAU.');
writeln
end;

procedure mess48;
begin
writeln;
writeln ('? VOULEZ VOUS MODIFIER UNE DONNEE INTRODUITE : (O/N)');
writeln
end;

procedure mess49(p : integer);
begin
writeln;
writeln ('# VOUS N''AVEZ PAS DECRIT TOUTES LES ENTREES DE LA PROCEDURE :',p:2);
writeln
end;

procedure mess49b;
begin
writeln;
writeln ('# VOTRE DESCRIPTION DES BRANCHES EST INCOHERENTE.');
writeln
end;

procedure mess50;
begin
writeln;
writeln ('? VOULEZ VOUS MODIFIER UNE DONNEE INTRODUITE : (O/N)');
writeln
end;

procedure mess51;
begin
writeln;
writeln (' VOULEZ VOUS MODIFIER LE RESEAU : (O/N)');
writeln
```

```
end;

procedure mess52;
begin
writeln;
writeln ('!-------------------------------------------------------------------------!');
writeln ('!                                                                         !');
writeln ('!                        AFFICHAGE DU RESEAU                              !');
writeln ('!                                                                         !');
writeln ('!-------------------------------------------------------------------------!');
writeln;
writeln;
writeln ('? DESIREZ VOUS AFFICHER LE RESEAU : (O/N)');
writeln
end;

procedure mess53(nom : elnt;
                 pe  : integer;
                 ps  : integer);
var i : integer;
begin
writeln;
writeln;
writeln;
write ('            SOUS RESEAU : ');
for i := 1 to bo2nt do write (nom[i]);
writeln;
writeln ('             ------------------');
writeln;
writeln;
writeln ('PROCEDURE D''ENTREE : ',pe:2,', PROCEDURE DE SORTIE : ',ps:2);
writeln;
writeln ('PROCEDURE SORTIE   PROCEDURE ENTREE ARCS');
end;

procedure mess54(num,k,proc,m : integer);
begin
write (num:6,'     ',k:4,' ----> ',proc:4,'     ',m:4,'    ');
end;

procedure mess55(ind : integer);
var t   : integer;
    arc : arcs;
begin
t := trunc(ind / 1000);
ind := ind - (t * 1000);
if t = 0
then arc := tnt[ind]
else if t = 1
     then arc := ttl[ind]
     else if t = 2
          then arc := ttv[ind]
          else arc := ttg[ind];
for ind := 1 to bo2nt do write(arc[ind]);
write(' ')
end;
```

```
PAGE    14    LIST VERSION  120281 3  11/ 8/82  15:44:37   CRERES

procedure mess56;
begin
writeln
end;

procedure helec1(borne : integer);
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU INITIALISATION DU RESEAU.');
writeln ('* VEUILLEZ ENTRER LE NOMBRE D''ELEMENTS QUE VOUS DISTINGUEZ DANS');
writeln ('* LA CLASSE DE MOTS QUE VOUS ETES EN TRAIN DE DECRIRE, CE NOMBRE');
writeln ('* DEVANT ETRE UN ENTIER INFERIEUR A ',borne:2,' :');
writeln
end;

procedure helec2(borne : integer);
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU INITIALISATION DU RESEAU.');
writeln ('* VEUILLEZ ENTRER LES ELEMENTS QUE VOUS DISTINGUEZ DANS LA CLASSE');
writeln ('* DE MOTS QUE VOUS ETES EN TRAIN DE DECRIRE. CES ELEMENTS DOIVENT');
writeln ('* AVOIR AU PLUS ',borne:2,' CARACTERES.');
writeln ('* S''ILS EN ONT PLUS, ON NE TIENDRA COMPTE QUE DES ',borne:2);
writeln ('* PREMIERS CARACTERES. LES ELEMENTS DOIVENT ETRE SEPARES PAR AU');
writeln ('* MOINS UN BLANC. TOUT AUTRE CARACTERE EST UN CARACTERE');
writeln ('* SIGNIFICATIF :');
writeln
end;

procedure helec3;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU INITIALISATION DU RESEAU.');
writeln ('* LE NOMBRE QUE VOUS AVEZ INDIQUE NE CORRESPOND PAS AU NOMBRE REEL');
writeln ('* D''ELEMENTS ENTRES. VOUS POUVEZ RECOMMENCER PAR ENTRER UN AUTRE');
writeln ('* NOMBRE (REPONDEZ OUI) ET REINTRODUIRE VOTRE LISTE D''ELEMENTS,');
writeln ('* OU BIEN NE REINTRODUIRE QUE LA LISTE DES ELEMENTS, LE NOMBRE IN-');
writeln ('* DIQUE RESTANT VALABLE (REPONDEZ NON) :');
writeln
end;

procedure heveri;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU INITIALISATION DU RESEAU.');
writeln ('* VOUS AVEZ MAINTENANT LA POSSIBILITE DE MODIFIER LES DONNEES');
writeln ('* INTRODUITES AVANT DE PASSER AU TRAITEMENT DU RESEAU PROPRE-');
writeln ('* MENT DIT. SI VOUS VOULEZ UNE MODIFICATION TAPEZ OUI, SINON');
writeln ('* TAPEZ NON :');
writeln
end;

procedure hemont;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU INITIALISATION DU RESEAU.');
writeln ('* SI VOUS VOULEZ MODIFIER LA DEFINITION DES NON-TERMINAUX, RE-');
```

```
PAGE    15    LIST VERSION  120281 3  11/ 8/82  15:44:37   CRERES

writeln ('* PONDEZ OUI SINON REPONDEZ NON :');
writeln
end;

procedure hemotv;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU INITIALISATION DU RESEAU.');
writeln ('* SI VOUS VOULEZ MODIFIER LA DEFINITION DES TERMINAUX VRAIS,');
writeln ('* REPONDEZ OUI SINON REPONDEZ NON :');
writeln
end;

procedure hemotl;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU INITIALISATION DU RESEAU.');
writeln ('* SI VOUS VOULEZ MODIFIER LA DEFINITION DES TERMINAUX LEXIQUES');
writeln ('* FIGES, REPONDEZ OUI SINON REPONDEZ NON :');
writeln
end;

procedure hemotg;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU INITIALISATION DU RESEAU.');
writeln ('* SI VOUS VOULEZ MODIFIER LA DEFINITION DES TERMINAUX GENERAUX,');
writeln ('* REPONDEZ OUI SINON REPONDEZ NON :');
writeln
end;

procedure henom;
var i,j : integer;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
writeln ('*        SOUS-NIVEAU INITIALISATION DU SOUS-RESEAU.');
writeln ('* VOUS DEVEZ MAINTENANT DECRIRE MAINTENANT SOUS-RESEAU APRES');
writeln ('* SOUS-RESEAU. POUR CELA VOUS DEVEZ INTRODUIRE UN NON-TERMINAL');
writeln ('* N''AYANT PAS ENCORE ETE DECRIT ET FIGURANT DANS L''ENSEMBLE');
writeln ('* DES NON-TERMINAUX QUE VOUS AVEZ DEFINI DANS LA PHASE INITIA-');
writeln ('* LISATION. LES NON-TERMINAUX DEJA DECRITS SONT :');
writeln ('*');
for i := 1 to bo1nt do
     begin
     if decrit[i]
     then begin
          write ('*   - ');
          for j := 1 to bo2nt do write(tnt[i,j]);
          writeln
          end
     end;
writeln ('*');
writeln ('* CONTINUEZ LA DESCRIPTION :');
writeln
end;
```

```
PAGE    16    LIST VERSION  120281 3  11/ 8/82  15:44:37   CRERES

procedure hepe;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
writeln ('*        SOUS-NIVEAU INITIALISATION DU SOUS-RESEAU.');
writeln ('* VOUS DEVEZ INDIQUER MAINTENANT QUEL EST LE NUMERO DE LA PRO-');
writeln ('* CEDURE QUI EST LA PROCEDURE D''ENTREE DANS LE SOUS-RESESAU QUE');
writeln ('* VOUS ETES EN TRAIN DE DECRIRE. CE NUMERO DOIT ETRE UN ENTIER :');
writeln
end;

procedure heps;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
writeln ('*        SOUS-NIVEAU INITIALISATION DU SOUS-RESEAU.');
writeln ('* VOUS DEVEZ INDIQUER MAINTENANT QUEL EST LE NUMERO DE LA PRO-');
writeln ('* CEDURE QUI EST LA PROCEDURE DE SORTIE DU SOUS-RESEAU QUE VOUS');
writeln ('* ETES EN TRAIN DE DECRIRE. CE NUMERO DOIT ETRE UN ENTIER :');
writeln
end;

procedure henbp;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
writeln ('*        SOUS-NIVEAU INITIALISATION DU SOUS-RESEAU.');
writeln ('* VOUS DEVEZ INDIQUER MAINTENANT LE NOMBRE DE PROCEDURES DU SOUS-');
writeln ('* RESESAU QUE VOUS ETES EN TRAIN DE DECRIRE. CE NOMBRE DOIT ETRE');
writeln ('* UN ENTIER :');
writeln
end;

procedure hemosr;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
writeln ('*        SOUS-NIVEAU INITIALISATION DU SOUS-RESEAU.');
writeln ('* VOUS AVEZ MAINTENANT LA POSSIBILITE DE MODIFIER LA DESCRIPTION');
writeln ('* DU SOUS-RESEAU DONT VOUS VENEZ DE DEFINIR UNE PARTIE. SI VOUS');
writeln ('* AVEZ UNE MODIFICATION A APPORTER, REPONDEZ OUI, SINON NON :');
writeln
end;

procedure henum;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
writeln ('*        SOUS-NIVEAU TRAITEMENT DES NOEUDS PROCEDURAUX.');
writeln ('* VOUS DEVEZ DECRIRE, UNE PAR UNE, LES PROCEDURES DU SOUS-RESEAU');
writeln ('* QUE VOUS ETES EN TRAIN DE DECRIRE. POUR CELA, COMMENCEZ PAR IN-');
writeln ('* TRODUIRE LE NUMERO DE LA PROCEDURE QUE VOUS DESIREZ DECRIRE. CE');
writeln ('* NUMERO DOIT ETRE UN ENTIER, DE PLUS, LA PREMIERE PROCEDURE DE-');
writeln ('* CRITE DOIT ETRE LA PROCEDURE D''ENTREE, TANDIS QUE LA DERNIERE');
writeln ('* DOIT ETRE LA PROCEDURE DE SORTIE :');
writeln
end;
```

```
PAGE    17    LIST VERSION  120281 3  11/ 8/82  15:44:37   CRERES

procedure henbe;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
writeln ('*        SOUS-NIVEAU TRAITEMENT DES NOEUDS PROCEDURAUX.');
writeln ('* VOUS DEVEZ INTRODUIRE, POUR LA PROCEDURE QUE VOUS AVEZ DECIDE');
writeln ('* DE DECRIRE, LE NOMBRE D''ARCS A L''ENTREE. CE NOMBRE DOIT ETRE');
writeln ('* UN ENTIER :');
writeln
end;

procedure henbs;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
writeln ('*        SOUS-NIVEAU TRAITEMENT DES NOEUDS PROCEDURAUX.');
writeln ('* VOUS DEVEZ INTRODUIRE, POUR LA PROCEDURE QUE VOUS AVEZ DECIDE');
writeln ('* DE DECRIRE, LE NOMBRE D''ARCS A LA SORTIE. CE NOMBRE DOIT ETRE');
writeln ('* UN ENTIER :');
writeln
end;

procedure herep1;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
writeln ('*                   GESTION DES REPRISES.');
writeln ('* VOUS AVEZ LA POSSIBILITE DE RECOMMENCER LA DESCRIPTION DU SOUS-');
writeln ('* RESEAU AU DEBUT, CAD OUBLIER TOUT CE QUE VOUS AVEZ FAIT A PRO-');
writeln ('* POS DE CE SOUS-RESEAU, DANS CE CAS REPONDEZ OUI; OU BIEN NE RE-');
writeln ('* COMMENCER QUE LA DESCRIPTION DE LA PROCEDURE OU DES PROCEDURES,');
writeln ('* DANS CE CAS, REPONDEZ NON :');
writeln
end;

procedure herep2;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
writeln ('*                   GESTION DES REPRISES.');
writeln ('* VOUS AVEZ LA POSSIBILITE DE REPRENDRE LA DESCRIPTION DU SOUS-');
writeln ('* RESEAU AU DEBUT DE LA DESCRIPTION DES PROCEDURES CONCERNANT LE');
writeln ('* SOUS-RESEAU EN COURS, CAD OUBLIER TOUTES LES DESCRIPTIONS DE');
writeln ('* PROCEDURES DE CE SOUS-RESEAU, DANS CE CAS REPONDEZ OUI; OU BIEN');
writeln ('* NE REPRENDRE QUE LA DESCRIPTION ERRONNEE, DANS CE CAS REPONDEZ');
writeln ('* NON. (DANS LE CAS OU VOUS AURIEZ OUBLIE LA DESCRIPTION DE LA');
writeln ('* PROCEDURE D''ENTREE OU DE LA PROCEDURE DE SORTIE, APRES AVOIR');
writeln ('* DECRIT TOUT LE SOUS-RESEAU, IL FAUT REPRENDRE LA DESCRIPTION');
writeln ('* DES PROCEDURES, CAD REPONDRE OUI) :');
writeln
end;

procedure herep3;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
```

```
PAGE    18    LIST VERSION  120281 3  11/ 8/82  15:44:37   CRERES

writeln ('*                    GESTION DES REPRISES.');
writeln ('* VOUS AVEZ LA POSSIBILITE DE MODIFIER LES DONNEES QUE VOUS VENEZ');
writeln ('* D''ENTRER. SI VOUS VOULEZ REVENIR EN ARRIERE, REPONDEZ OUI, SI-');
writeln ('* NON REPONDEZ NON :');
writeln
end;

procedure hebr1;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
writeln ('*        SOUS-NIVEAU TRAITEMENT DES BRANCHES LINEAIRES.');
writeln ('* VOUS DEVEZ INTRODUIRE MAINTENANT LE NUMERO DE L''ENTREE DE LA');
writeln ('* PROCEDURE D''ARRIVEE DE LA BRANCHE CORRESPONDANT A LA SORTIE');
writeln ('* INDIQUEE, BRANCHE ABOUTISSANT A CETTE ENTREE :');
writeln
end;

procedure hebr2;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
writeln ('*        SOUS-NIVEAU TRAITEMENT DES BRANCHES LINEAIRES.');
writeln ('* VOUS DEVEZ INTRODUIRE MAINTENANT LE NUMERO DE LA PROCEDURE');
writeln ('* D''ARRIVEE DE LA BRANCHE QUE VOUS VENEZ DE DECRIRE :');
writeln
end;

procedure hearc(boarc : integer);
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
writeln ('*        SOUS-NIVEAU TRAITEMENT DES BRANCHES LINEAIRES.');
writeln ('* VOUS DEVEZ DECRIRE MAINTENANT LES ARCS QUI CONSTITUENT LA');
writeln ('* BRANCHE, CAD INTRODUIRE LES NOMS DES ARCS, SEPARES PAR AU MOINS');
writeln ('* UN BLANC. IL PEUT Y AVOIR AU PLUS : ',boarc:2,' ARCS PAR ');
writeln ('* BRANCHE :');
writeln
end;

procedure herep4;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
writeln ('*                    GESTION DES REPRISES.');
writeln ('* VOUS AVEZ MAINTENANT LA POSSIBILITE DE REVENIR EN ARRIERE ET');
writeln ('* D''OUBLIER TOUT CE QUI CONCERNE LA DESCRIPTION DES BRANCHES');
writeln ('* ET DE RECOMMENCER LE TRAITEMENT A UN NIVEAU ANTERIEUR.');
writeln ('* SI VOUS VOULEZ OUBLIER CETTE DESCRIPTION,REPONDEZ OUI, SINON');
writeln ('* REPONDEZ NON :');
writeln
end;

procedure herep5;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
```

```
PAGE    19    LIST VERSION  120281 3  11/ 8/82  15:44:37   CRERES

writeln ('*                        GESTION DES REPRISES.');
writeln ('* VOUS AVEZ MAINTENANT LA POSSIBILITE DE REVENIR EN ARRIERE ET');
writeln ('* D''OUBLIER TOUTES LES DESCRIPTIONS DE BRANCHES CONCERNANT LE');
writeln ('* SOUS-RESEAU EN COURS ET DE RECOMMENCER CES DESCRIPTIONS. SI');
writeln ('* VOUS DESIREZ RECOMMENCER, REPONDEZ OUI, SINON REPONDEZ NON :');
writeln
end;

procedure herep7;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
writeln ('*                        GESTION DES REPRISES.');
writeln ('* VOUS AVEZ LA POSSIBILITE DE MODIFIER LES DONNEES QUE VOUS VENEZ');
writeln ('* D''ENTRER A PROPOS DU DERNIER SOUS-RESEAU DECRIT. SI VOUS VOU-');
writeln ('* LEZ REVENIR EN ARRIERE, REPONDEZ OUI, SINON REPONDEZ NON :');
writeln
end;

procedure herep8;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU TRAITEMENT DU RESEAU.');
writeln ('*                        GESTION DES REPRISES.');
writeln ('* VOUS AVEZ LA POSSIBILITE DE MODIFIER LE RESEAU. CETTE DEMANDE');
writeln ('* ENTRAINE CEPENDANT L''EFFACEMENT DE TOUT CE QUI A ETE FAIT PEN-');
writeln ('* DANT LA PHASE DE TRAITEMENT DU RESEAU, VOUS REPRENEZ DONC DE-');
writeln ('* PUIS LE DEBUT. SI VOUS VOULEZ MODIFIER, REPONDEZ OUI, SINON,');
writeln ('* REPONDEZ NON :');
writeln
end;

procedure heimp;
begin
writeln;
writeln ('*        VOUS ETES AU NIVEAU AFFICHAGE DU RESEAU.');
writeln ('* VOUS POUVEZ MAINTENANT FAIRE AFFICHER LE RESEAU A L''ECRAN');
writeln ('* POUR CONTROLE VISUEL. POUR AFFICHER,REPONDEZ OUI, SINON RE-');
writeln ('* PONDEZ NON :');
writeln
end;

(*****************************************************************************)
(*                                                                           *)
(*                          PROCEDURES UTILITAIRES                           *)
(*                                                                           *)
(* CES PROCEDURES SONT UTILITAIRES EN CE SENS QU'ELLES SONT APPELEES DE       *)
(* SIEURS ENDROITS DIFFERENTS DANS LE CODE. ELLES SUPPOSENT CEPENDANT        *)
(* TOUTES LA DEFINITION DE CERTAINS TYPES GLOBAUX OU DE CERTAINES VARIA-     *)
(* BLES GLOBALES. UNE MODIFICATION DE CES TYPES OU DE CES VARIABLES EN-      *)
(* TRAINE UNE MODIFICATION DES PROCEDURES OU FONCTIONS LES UTILISANT.        *)
(*                                                                           *)
(*****************************************************************************)

procedure inint(var nombre : integer;
                var code   : codret);
(*****************************************************************************)
```

```
PAGE    20    LIST VERSION  120281 3  11/ 8/82  15:44:37   CRERES

(*                                                                    *)
(*                  LA PROCEDURE ININT                                *)
(*                                                                    *)
(* La procedure ININT lit un entier au terminal. Si cette lecture     *)
(* a reussi, cet entier se retrouve dans l'argument 'nombre'. Si-     *)
(* non, ce dernier a une valeur indeterminee. La reussite de la       *)
(* lecture est indiquee par l'argument 'code' :                       *)
(*                                                                    *)
(*       code = blank   ==>  l'utilisateur n'a rien tape              *)
(*       code = help    ==>  l'utilisateur veut des precisions        *)
(*       code = invalid ==>  l'utilisateur a tape qch d'insignifiant  *)
(*       code = ok      ==>  la lecture a reussi                      *)
(*                                                                    *)
(* Cette procedure suppose la predefinition du type codret.           *)
(*                                                                    *)
(**********************************************************************)
var cas    : 0..4;
    ch     : char;
begin
nombre := 0;
cas := 0;
while not eoln(input) do
     begin
     read(ch);
     if (ch = ' ')
     then if cas = 3
          then cas := 4;
     if (ch = '?')
     then if cas = 0
          then cas := 1
          else cas := 2;
     if ((ch <> '?') and (ch <> ' ') and
         ((ord('0') > ord(ch)) or (ord('9') < ord(ch))))
     then cas := 2;
     if ((ord('0') <= ord(ch))and (ord('9') >= ord(ch)))
     then if ((cas = 0) or (cas = 3))
          then begin
               cas := 3;
               nombre := (nombre * 10) + (ord(ch) - ord('0'))
               end
          else cas := 2
      end;
readln;
case cas of
     0 : code := blank;
     1 : code := help;
     2 : code := invalid;
     3 : code := ok;
     4 : code := ok;
     end
end;

procedure inoui(var reponse : char;
                var code    : codret);
(**********************************************************************)
(*                                                                    *)
(*                  LA PROCEDURE INOUI                                *)
```

```
PAGE    21    LIST VERSION  120281 3  11/ 8/82  15:44:37   CRERES

(*                                                                    *)
(* La procedure INOUI lit une reponse au terminal qui doit soit      *)
(* etre oui soit non.                                                *)
(* Est reconnu comme oui : o,ou,oui;                                 *)
(* est reconnu comme non : n,no,non.                                 *)
(* La reponse se trouve dans l'argument 'reponse' si la lecture      *)
(* a reussi. La valeur de l'argument est 'o' dans le cas oui et      *)
(* 'n' dans le cas non. L'argument code indique si la lecture a      *)
(* reussi :                                                          *)
(*                                                                   *)
(*       code = blank   ==> l'utilisateur n'a rien tape              *)
(*       code = help    ==> l'utilisateur veut des precisions        *)
(*       code = invalid ==> l'utilisateur a tape qch d'insignifiant  *)
(*       code = ok      ==> la lecture a reussi                      *)
(*                                                                   *)
(* La procedure suppose la predefinition du type codret.             *)
(*                                                                   *)
(*********************************************************************)
var cas    : 0..10;
    ch     : char;
begin
cas := 0;
while not eoln(input) do
     begin
     read(ch);
     if ch = ' '
     then if cas = 2
          then cas := 3
          else if cas = 4
               then cas := 10
               else if cas = 6
                    then cas := 7
                    else if cas = 8
                         then cas := 10;
     if ch = '?'
     then if cas = 0
          then cas := 1
          else cas := 10;
     if ((ch = 'N') or (ch = 'n'))
     then if cas = 0
          then cas := 6
          else if cas = 8
               then cas := 9
               else cas := 10;
     if ((ch = '0') or (ch = 'o'))
     then if cas = 0
          then cas := 2
          else if cas = 6
               then cas := 8
               else cas := 10;
     if ((ch = 'U') or (ch = 'u'))
     then if cas = 2
          then cas := 4
          else cas := 10;
     if ((ch = 'I') or (ch = 'i'))
     then if cas = 4
          then cas := 5
```

```
PAGE    22    LIST VERSION  120281 3  11/ 8/82  15:44:37   CRERES

          else cas := 10;
     if ((ch <> 'O') and (ch <> 'o') and
         (ch <> 'N') and (ch <> 'n') and
         (ch <> 'U') and (ch <> 'u') and
         (ch <> 'I') and (ch <> 'i') and
         (ch <> '?') and (ch <> ' '))
     then cas := 10
     end;
readln;
case cas of
     0       : code := blank;
     1       : code := help;
     2,3,4,5 : begin
               code := ok;
               reponse := 'o'
               end;
     6,7,8,9 : begin
               code := ok;
               reponse := 'n'
               end;
    10       : code := invalid;
    end
end;

procedure lecture(borne1     : integer;
                  borne2     : integer;
                  var nombre : integer;
                  var liste  : tableau);
(*************************************************************************)
(*                                                                       *)
(*                    LA PROCEDURE LECTURE                               *)
(*                                                                       *)
(* La procedure LECTURE lit des elements au terminal et les memorise dans *)
(* l'argument liste. De plus elle lit le nombre d'elements introduits au  *)
(* terminal. Ces lectures se font jusqu'a ce que les donnees memorisees   *)
(* soient coherentes. La memorisation dans 'liste' se fait de telle ma-   *)
(* niere, qu'apres lecture, les elements puissent etre transferes, carac- *)
(* tere pae caractere, dans un tableau a deux dimansions, de bornes :     *)
(* 'borne1' et 'borne2'. 'borne1' et 'borne2' sont fournis et les resul-  *)
(* tats des lectures se trouvent dans 'nombre' et 'liste'.                *)
(*                                                                       *)
(*                    borne1 = nombre maximal d'elements                 *)
(*                    borne2 = nombre maximal de caracteres pris en      *)
(*                             compte par element                        *)
(*                                                                       *)
(* La procedure suppose la predifinition du type tableau et des proce-    *)
(* dures inoui, inint, vidint, invint, vidoui, invoui, mess1, helec1,     *)
(* mess2, mess3, mess4, mess5, mess6, helec2, mess7, helec3, mess8.       *)
(*                                                                       *)
(*************************************************************************)
var   ch           : char;
      code         : codret;
      reponse      : char;
      i,j          : integer;
      cas          : 0..3;
      incoherence  : boolean;
      nonchiffre   : boolean;
```

```
      nonelement  : boolean;
      repinvalide : boolean;

begin
incoherence := true;
while incoherence do
     begin
     mess1;
     nonchiffre := true;
     while nonchiffre do
          begin
          inint(nombre,code);
          case code of
          blank   : vidint;
          help    : helec1(borne1);
          invalid : invint;
          ok      : if nombre > borne1
                    then mess2(borne1)
                    else if nombre = 0
                         then mess3
                         else nonchiffre := false
                    end
          end;
     mess4;
     nonelement := true;
     while nonelement do
          begin
          for i := 1 to borne1 do
               begin
               for j := 1 to borne2 do
                    begin
                    liste [(i-1)*borne2+j] := ' '
                    end
               end;
          i := 1;
          j := 1;
          cas := 0;
          while (not eoln(input)) and (not (cas = 3)) do
               begin
               read (ch);
               if (ch = '?')
               then if cas = 0
                    then cas := 1
                    else cas := 2;
               if ((ch <> ' ') and (ch <> '?'))
                    then cas := 2;
               if (i > borne1) and (ch <> ' ')
               then begin
                    mess5(borne1);
                    i := i + 1;
                    cas := 3
                    end
               else begin
                    if (j = 1)
                    then begin
                         if (ch <> ' ')
                         then begin
```

```
                        liste [(i-1)*borne2+j] := ch;
                        j := j + 1
                        end
                   end
              else if ((1 < j) and (j <= borne2))
                   then if (ch = ' ')
                        then begin
                             for j := j to borne2 do
                             liste [(i-1)*borne2+j] := ' ';
                             i := i + 1;
                             j := 1
                             end
                        else begin
                             liste [(i-1)*borne2+j] := ch;
                             j := j + 1
                             end
                   else if (ch = ' ')
                        then begin
                             i := i + 1;
                             j := 1
                             end
              end
        end;
   readln;
   if j > 1 then i := i+1;
   case cas of
        0 : mess6;
        1 : helec2(borne2);
     2,3 : if (nombre <> (i - 1))
           then begin
                mess7;
                repinvalide := true;
                while repinvalide do
                     begin
                     inoui(reponse,code);
                     case code of
                     blank   : vidoui;
                     help    : helec3;
                     invalid : invoui;
                     ok      : if reponse = 'o'
                               then begin
                                    nonelement := false;
                                    repinvalide := false
                                    end
                               else begin
                                    repinvalide := false;
                                    mess8
                                    end;
                               end
                     end
                end
           else begin
                nonelement := false;
                incoherence := false;
                end
        end
   end
```

```
    end
end;

procedure rech(    nom : chaine;
                var ind : integer);
(*************************************************************************)
(*                                                                       *)
(*                  LA PROCEDURE RECH                                    *)
(*                                                                       *)
(* La procedure RECH recherche un element donne par l'argument 'chaine'  *)
(* dans les tableaux tnt, ttl, ttv et ttg. Si l'element ne se trouve dans *)
(* aucun de ces tableaux, ind a la valeur 0. Sinon ind a la valeur de l'in- *)
(* dice pointant vers l'element dans le tableau, augmente de :           *)
(*                                                                       *)
(*                  - 0    si l'element se trouve dans tnt               *)
(*                  - 1000 si l'element se trouve dans ttl               *)
(*                  - 2000 si l'element se trouve dans ttv               *)
(*                  - 3000 si l'element se trouve dans ttg               *)
(*                                                                       *)
(* La procedure suppose la predefinition des variables nbnt, nbtv, nbtl  *)
(* nbtg, tnt, ttv, ttl et ttg.                                           *)
(*                                                                       *)
(*************************************************************************)
var i : integer;
begin
ind := 0;
for i := 1 to nbnt do
if nom = tnt[i] then ind := i;
if ind = 0 then
for i := 1 to nbtv do
if nom = ttv[i] then ind := i + 2000;
if ind = 0 then
for i := 1 to nbtl do
if nom = ttl[i] then ind := i + 1000;
if ind = 0 then
for i := 1 to nbtg do
if nom = ttg[i] then ind := i + 3000
end;

procedure instring(var nom  : chaine;
                   var code : codret);
(*************************************************************************)
(*                                                                       *)
(*                  LA PROCEDURE INSTRING                                *)
(*                                                                       *)
(* La procedure INSTRING lit une chaine de caracteres au terminal. Cette *)
(* chaine peut se trouver n'importe ou sur la ligne, elle ne peut cependant *)
(* renfermer de caractere blanc. Dans la cas d'une lecture avec succes,  *)
(* l'argument 'chaine' contiendra les bo2nt premiers caracteres de la    *)
(* chaine lue. Le succes de la lecture est indique par l'argument code :  *)
(*                                                                       *)
(*       code = blank   ===> l'utilisateur n'a rien tape                 *)
(*       code = help    ===> l'utilisateur veut des precisions           *)
(*       code = invalid ===> l'utilisateur a tape qch d'insignifiant     *)
(*       code = ok      ===> la lecture a reussi                         *)
(*                                                                       *)
(* La procedure suppose la predefinition des types chaine et codret, et de *)
```

```
(* la constante bo2nt.                                                    *)
(*                                                                        *)
(**************************************************************************)
var i   : integer;
    ch  : char;
    cas : 1..6;
begin
for i := 1 to bo2nt do nom[i] := ' ';
cas := 1;
i := 1;
while not eoln(input) and (not (cas = 6)) do
     begin
     read(ch);
     if ch = ' '
     then if cas = 2
          then cas := 3
          else if cas = 4
               then cas := 5;
     if ch = '?'
     then if cas = 1
          then cas := 2
          else if (cas = 2) or (cas = 4)
               then cas := 4
               else cas := 6;
     if (ch <> ' ') and (ch <> '?')
     then if (cas = 1) or (cas = 2) or (cas = 4)
          then cas := 4
          else cas := 6;
     if (ch <> ' ') and (i <= bo2nt)
     then begin
          nom[i] := ch;
          i := i + 1
          end
     end;
readln;
case cas of
     1   : code := blank;
     2,3 : code := help;
     4,5 : code := ok;
     6   : code := invalid
     end
end;

procedure inarc(var tarc : tablarc;
                var code : codret);
(**************************************************************************)
(*                                                                        *)
(*                    LA PROCEDURE INARC                                  *)
(*                                                                        *)
(* La procedure INARC lit une chaine de caracteres au terminal. Cette     *)
(* chaine peut comporter des blancs. Chaque suite de caracteres de cette  *)
(* chaine ne renfermant pas de caractere blanc est interpretee comme etant*)
(* un element. Les elements sont memorises dans le tableau tarc, en ne con-*)
(* derant que les bo2nt premiers caracteres de chaque element. Si la lec- *)
(* ture a reussie, tarc contient les elements. Le succes de la lecture est*)
(* indique par l'argument code :                                          *)
(*                                                                        *)
```

```
PAGE    27    LIST VERSION  120281 3  11/ 8/82  15:44:37   CRERES

(*        code = blank   ===> l'utilisateur n'a rien tape                  *)
(*        code = help    ===> l'utilisateur demande des precisions         *)
(*        code = invalid ===> l'utilisateur a tape qch d'insignifiant      *)
(*        code = ok      ===> la lecture a reussi                          *)
(*                                                                         *)
(* La procedure suppose la predefinition des types tablarc, codret et arcs, *)
(* des constantes boarc et bo2nt et des procedures mess9 et mess10.        *)
(*                                                                         *)
(*************************************************************************)
var arc    : arcs;
    ch     : char;
    cas    : 1..5;
    i,j,k  : integer;
    ind    : integer;
begin
cas := 1;
for i := 1 to boarc do for j := 1 to bo2nt do tarc[i,j] := ' ';
i := 1;
j := 1;
while (not eoln(input)) and (not (cas = 5)) do
     begin
     read (ch);
     if (ch = ' ') and (cas = 2)
     then cas := 3;
     if (ch = '?')
     then if cas = 1
          then cas := 2
          else if (cas = 2) or (cas = 3)
               then cas := 4;
     if (ch <> ' ') and (ch <> '?')
     then if (cas = 1) or (cas = 2) or (cas = 3)
          then cas := 4;
     if (i > boarc) and (ch <> ' ')
     then begin
          cas := 5;
          mess9(boarc)
          end
     else if ch = ' '
          then if (j > 1) and (j <= bo2nt)
               then begin
                    for j := j to bo2nt do tarc[i,j] := ' ';
                    j := 1;
                    i := i + 1
                    end
               else begin
                    if j > bo2nt
                    then begin
                         j := 1;
                         i := i + 1
                         end
                    end
          else if j <= bo2nt
               then begin
                    tarc[i,j] := ch;
                    j := j + 1
                    end
     end;
```

```
readln;
if cas = 4
then begin
     if (j <= bo2nt) and (1 < j)
     then begin
          for j := j to bo2nt do tarc[i,j] := ' ';
          i := i + 1
          end
     else if j > bo2nt then i := i + 1;
     for j := 1 to (i-1) do
          begin
          arc := tarc[j];
          rech(arc,ind);
          if ind <= 0
          then begin
               mess10(arc);
               cas := 5
               end
          end
     end;
case cas of
     1    : code := blank;
     2,3  : code := help;
     4    : code := ok;
     5    : code := invalid
     end
end;

(*************************************************************************)
(*                                                                       *)
(*                  LES FONCTIONS AD ET INDT                             *)
(*                                                                       *)
(* La fonction AD recherche un element k dans un tableau tab contenant nbp  *)
(* elements de 5 composantes. Si k figure parmi une des premieres compo-    *)
(* santes des elements du tableau, AD prend la valeur du deuxieme compo-    *)
(* sant correspondant a cet element. Elle suppose la predefinition du type  *)
(* sauvproc.                                                                *)
(* La fonction INDT a une fonction analogue, sauf qu'elle prend la valeur   *)
(* de l'indice quand l'element figure dans le tableau. Elle suppose egale-  *)
(* ment la predefinition du type sauvproc.                                  *)
(*                                                                       *)
(*************************************************************************)
function ad (k   : integer;
             tab : sauvproc;
             nbp : integer):integer;
var i : integer;
begin
ad := 0;
for i := 1 to nbp do if tab[i,1] = k then ad := tab[i,2]
end;

function indt (num : integer;
               tab : sauvproc;
               nbp : integer): integer;
var i : integer;
begin
indt := 0;
```

```
for i := 1 to nbp do if tab[i,1] = num then indt := i
end;

(*****************************************************************************)
(*                                                                           *)
(*                              AUTRES PROCEDURES                            *)
(*                                                                           *)
(* CES PROCEDURES ONT ETE SEPAREES DU PROGRAMME PRINCIPAL, AFIN DE METTRE    *)
(* EN EVIDENCE LA STRUCTURE DU MODULE CENTRAL ET DE RENDRE LE CODE PLUS LI-  *)
(*    SIBLE                                                                  *)
(*                                                                           *)
(*****************************************************************************)

(*****************************************************************************)
(*                                                                           *)
(*                          PROCEDURES D'INITIALISATION                      *)
(*                                                                           *)
(*****************************************************************************)

procedure initnt;
(* on decrit les non-terminaux *)
begin
mess11;
lecture (bo1nt,bo2nt,nbnt,tab);
for i:=1 to bo1nt do for j:=1 to bo2nt do tnt[i,j]:=tab[(i-1)*bo2nt+j];
end;

procedure inittv;
(* on decrit les terminaux vrais *)
begin
mess12;
lecture (bo1tv,bo2tv,nbtv,tab);
for i:=1 to bo1tv do for j:=1 to bo2tv do ttv[i,j]:=tab[(i-1)*bo2tv+j];
end;

procedure inittl;
(* on decrit les terminaux lexiques figes *)
begin
mess13;
lecture (bo1tl,bo2tl,nbtl,tab);
for i:=1 to bo1tl do for j:=1 to bo2tl do ttl[i,j]:=tab[(i-1)*bo2tl+j];
end;

procedure inittg;
(* on decrit les terminaux generaux *)
begin
mess14;
lecture (bo1tg,bo2tg,nbtg,tab);
for i:=1 to bo1tg do for j:=1 to bo2tg do ttg[i,j]:=tab[(i-1)*bo2tg+j];
end;

procedure modini;
begin
mess16;
repinvalide := true;
while repinvalide do
     begin
```

```
      inoui (reponse,code);
      case code of
           blank   : vidoui;
           help    : heveri;
           invalid : invoui;
           ok      : repinvalide := false
           end
      end;
if reponse = 'o'
then begin
      mess17;
      repinvalide := true;
      while repinvalide do
           begin
           inoui (reponse,code);
           case code of
                blank   : vidoui;
                help    : hemont;
                invalid : invoui;
                ok      : repinvalide := false
                end
           end;
      if reponse = 'o' then initnt;
      mess18;
      repinvalide := true;
      while repinvalide do
           begin
           inoui (reponse,code);
           case code of
                blank   : vidoui;
                help    : hemotv;
                invalid : invoui;
                ok      : repinvalide := false
                end
           end;
      if reponse = 'o' then inittv;
      mess19;
      repinvalide := true;
      while repinvalide do
           begin
           inoui (reponse,code);
           case code of
                blank   : vidoui;
                help    : hemotl;
                invalid : invoui;
                ok      : repinvalide := false
                end
           end;
      if reponse = 'o' then inittl;
      mess20;
      repinvalide := true;
      while repinvalide do
           begin
           inoui (reponse,code);
           case code of
                blank   : vidoui;
                help    : hemotg;
```

```
PAGE    31    LIST VERSION  120281 3  11/ 8/82  15:44:37   CRERES

                invalid : invoui;
                ok      : repinvalide := false
                end
          end;
     if reponse = 'o' then inittg;
     mess15;
     reponse := 'o'
     end
end;

(*************************************************************************)
(*                                                                       *)
(*                  PROCEDURES DE GESTION DES REPRISES                   *)
(*                                                                       *)
(*************************************************************************)

procedure repno;
(*****************************)
(* REPRISE AU NIVEAU DU NOEUD *)
(*****************************)
begin
reprsr := false;
mess21;
repinvalide := true;
while repinvalide do
     begin
     inoui(reponse,code);
     case code of
          blank   : vidoui;
          help    : herep1;
          invalid : invoui;
          ok      : repinvalide := false
          end
     end;
if reponse = 'o'
then begin
     nincor := true;
     dsrincor := true;
     reprsr := true;
     decrit[ind] := false;
     for k := (j-i) downto 1 do
          begin
          adc := adc-table[j,3]-table[j,4]-1;
          reseau[adc+table[j,3]] := 0
          end
     end
else begin
     mess22;
     repinvalide := true;
     while repinvalide do
          begin
          inoui(reponse,code);
          case code of
               blank   : vidoui;
               help    : herep2;
               invalid : invoui;
               ok      : repinvalide := false
```

```
PAGE    32    LIST VERSION  120281 3  11/ 8/82  15:44:37   CRERES

                end
           end;
      if reponse = 'o'
      then begin
           nincor := true;
           for k := (j-1) downto 1 do
                begin
                adc:=adc-table[j,3]-table[j,4]-1;
                reseau[adc+table[j,3]] := 0
                end;
           j := 1
           end
      end
end;

procedure repbr;
(*********************************)
(* REPRISE AU NIVEAU DES BRANCHES *)
(*********************************)
begin
reprsr := false;
reprbr := false;
mess23;
repinvalide := true;
while repinvalide do
      begin
      inoui(reponse,code);
      case code of
           blank   : vidoui;
           help    : herep4;
           invalid : invoui;
           ok      : repinvalide := false
           end
      end;
if reponse = 'o'
then begin
      reprsr := true;
      reprbr := true
      end
else begin
      mess24;
      repinvalide := true;
      while repinvalide do
           begin
           inoui(reponse,code);
           case code of
                blank   : vidoui;
                help    : herep5;
                invalid : invoui;
                ok      : repinvalide := false
                end
           end;
      if reponse = 'o'
      then reprbr := true
      end;
 if reprbr
then begin
```

```
      k := k - 1;
      for l := j downto 1 do
            begin
            for m := k downto 1 do
                  begin
                  adc := adc -1;
                  reseau[-reseau[adc]] := 0;
                  reseau[adc] := 0;
                  adc := adc -1;
                  while reseau[adc] >= 0 do
                        begin
                        reseau[adc] := 0;
                        adc := adc - 1
                        end;
                  reseau[-reseau[adc]] := 0;
                  reseau[adc] := 0
                  end;
            if l > 1
            then k := table[l-1,4]
            end;
      for j := 1 to nbp do table[j,5] := table[j,3];
      j := 0
      end;
if reprsr
then begin
      j := nbp + 1;
      repno;
      if reponse = 'n'
      then begin
            reprsr := false;
            j := 0
            end
      else reprsr := true
      end
end;

(*************************************************************************)
(*                                                                       *)
(*              PROCEDURES DE TRAITEMENT DES SOUS-RESEAUX                *)
(*                                                                       *)
(*************************************************************************)

procedure trdsr;
(*********************************)
(* INITIALISATION DU SOUS-RESEAU *)
(*********************************)
begin
mess25(i);
nomincor := true;
while nomincor do
      begin
      repinvalide := true;
      while repinvalide do
            begin
            instring(nom,code);
            case code of
                  blank   : vidstring;
```

```
                help    : henom;
                invalid : invstring;
                ok      : repinvalide := false
                end
          end;
     rech(nom,ind);
     if (ind <= 0) or (ind > 1000)
     then mess26
     else if decrit[ind]
          then mess27
          else begin
                decrit[ind] := true;
                nomincor := false
                end
     end;
mess28;
repinvalide := true;
while repinvalide do
     begin
     inint(pe,code);
     case code of
          blank   : vidint;
          help    : hepe;
          invalid : invint;
          ok      : repinvalide := false
          end
     end;
mess29;
repinvalide := true;
while repinvalide do
     begin
     inint(ps,code);
     case code of
          blank   : vidint;
          help    : heps;
          invalid : invint;
          ok      : if ps = pe
                    then mess29b
                    else repinvalide := false
          end
     end;
mess30;
repinvalide := true;
while repinvalide do
     begin
     inint(nbp,code);
     case code of
          blank   : vidint;
          help    : henbp;
          invalid : invint;
          ok      : if nbp <= 1
                    then mess31
                    else if nbp > nproc
                         then mess31b
                         else repinvalide := false
          end
     end;
```

```
mess32;
repinvalide := true;
while repinvalide do
     begin
     inoui(reponse,code);
     case code of
          blank   : vidoui;
          help    : hemosr;
          invalid : invoui;
          ok      : repinvalide := false
          end
     end;
if reponse = 'n'
then dsrincor := false
else decrit[ind] := false
end;

procedure trno;
(*************************************)
(* TRAITEMENT DES NOEUDS-PROCEDURAUX *)
(*************************************)
begin
mess33(j);
repinvalide := true;
while repinvalide do
     begin
     inint(num,code);
     case code of
          blank   : vidint;
          help    : henum;
          invalid : invint;
          ok      : repinvalide := false
          end
     end;
if (j = 1) and (num <> pe)
then begin
     mess38b;
     erreur := true
     end
else if (j = nbp) and (num <> ps)
     then begin
          mess38c;
          erreur := true
          end
     else if (j <> 1) and (j <> nbp) and ((num = pe) or (num = ps))
          then begin
               mess38d;
               erreur := true
               end
          else begin
               k := j-1;
               if indt(num,table,k) <> 0
               then begin
                    erreur := true;
                    mess34
                    end
               else begin
```

```
                    erreur := false;
                    mess35;
                    repinvalide := true;
                    while repinvalide do
                         begin
                         inint(nbe,code);
                         case code of
                              blank   : vidint;
                              help    : henbe;
                              invalid : invint;
                              ok      : repinvalide := false
                              end
                         end;
                    if (((nbe = 0) and (num <> pe)) or
                        ((nbe <> 0) and (num = pe)))
                    then begin
                         erreur := true;
                         mess36
                         end
                    else begin
                         repinvalide := true;
                         mess37;
                         while repinvalide do
                              begin
                              inint(nbs,code);
                              case code of
                                   blank   : vidint;
                                   help    : henbs;
                                   invalid : invint;
                                   ok      : repinvalide := false
                                   end
                              end;
                         if (((nbs = 0) and (num <> ps)) or
                             ((nbs <> 0) and (num = ps)))
                         then begin
                              erreur := true;
                              mess38
                              end
                         end
               end
          end;
     if not erreur
     then begin
          table[j,1] := num;
          table[j,2] := adc + nbe;
          table[j,3] := nbe;
          table[j,4] := nbs;
          table[j,5] := nbe;
          reseau[adc+nbe] := -num;
          nnnn := trunc(num/10);
          if num >= 50
          then reseau[adc+nbe] := -(num-nnnn*10);
          adc := adc + nbe + nbs +1;
          j := j + 1
          end
     else repno
     end;
```

```
procedure trbr;
(*************************************)
(* TRAITEMENT DES BRANCHES LINEAIRES *)
(*************************************)
begin
num := table[j,1];
adr := table[j,2];
nbe := table[j,3];
nbs := table[j,4];
mess39(j,num);
k := 1;
reprbr := false;
while (k <= nbs) and (not reprbr) do
     begin
     mess40(k);
     repinvalide := true;
     while repinvalide do
          begin
          inint(ent,code);
          case code of
               blank   : vidint;
               help    : hebr1;
               invalid : invint;
               ok      : repinvalide := false
               end
          end;
     mess41;
     repinvalide := true;
     while repinvalide do
          begin
          inint(proc,code);
          case code of
               blank   : vidint;
               help    : hebr2;
               invalid : invint;
               ok      : repinvalide := false
               end
          end;
     indproc := indt(proc,table,nbp);
     if indproc = 0
     then begin
          mess42;
          erreur := true
          end
     else if table[indproc,3] < ent
          then begin
               mess44(proc,ent);
               erreur := true
               end
          else if table[indproc,5] <= 0
               then begin
                    mess45(proc);
                    erreur := true
                    end
               else if reseau[ad(proc,table,nbp)-ent] <> 0
                    then begin
```

```
                        mess46;
                        erreur := true
                        end
                    else erreur := false;
    if not erreur
    then begin
         mess46b;
         repinvalide := true;
         while repinvalide do
              begin
              inarc(tarc,code);
              case code of
                   blank   : repinvalide := false;
                   help    : hearc(boarc);
                   invalid : begin
                             repinvalide := false;
                             erreur := true
                             end;
                   ok      : repinvalide := false
                   end
              end
         end;
    if not erreur
    then begin
         reseau[adr+k] := adc + 1;
         reseau[adc] := -adr-k;
         adc := adc + 1;
         l := 1;
         arc := tarc[1];
         repeat
              begin
              rech(arc,indarc);
              reseau[adc] := indarc;
              adc := adc + 1;
              l := l + 1;
              if l <= boarc then arc := tarc[l]
              end
         until (arc[1] = ' ') or (l > boarc);
         reseau[adc] := -ad(proc,table,nbp) + ent;
         adc := adc + 1;
         reseau[ad(proc,table,nbp)-ent] := adc - 2;
         table[indproc,5] := table[indproc,5] - 1;
         k := k + 1
         end
    else repbr
    end;
j := j + 1;
end;

(*****************************************************************************)
(*                                                                           *)
(*                  PROCEDURES DE VERIFICATION DE COHERENCE                  *)
(*                                                                           *)
(*****************************************************************************)

procedure verno;
(*********************************)
```

```
(* COHERENCE AU NIVEAU DU NOEUD *)
(*********************************)
begin
k := 0;
l := 0;
for j := 1 to nbp do
     begin
     k := k + table[j,3];
     l := l + table[j,4]
     end;
if k <> l
then begin
     reponse := 'n';
     while reponse = 'n' do
          begin
          mess47;
          repno
          end
     end
else begin
     mess48;
     repinvalide := true;
     while repinvalide do
          begin
          inoui (reponse,code);
          case code of
               blank   : vidoui;
               help    : herep3;
               invalid : invoui;
               ok      : repinvalide := false
               end
          end;
     if reponse = 'n'
     then nincor := false
     else begin
          repno;
          if reponse = 'n' then nincor := false
          end
     end
end;

procedure verbr;
(*************************************)
(* COHERENCE AU NIVEAU DES BRANCHES *)
(*************************************)
begin
erreur := false;
for l := 1 to nbp do
     if table[l,5] > 0
     then begin
          erreur := true;
          mess49(table[l,1])
          end;
if erreur
then begin
     reponse := 'n';
     while reponse = 'n' do
```

```
          begin
          mess49b;
          k := 1;
          repbr
          end
     end
else begin
     mess50;
     repinvalide := true;
     while repinvalide do
          begin
          inoui(reponse,code);
          case code of
               blank    : vidoui;
               help     : herep7;
               invalid  : invoui;
               ok       : repinvalide := false
               end
          end;
     if reponse = 'n'
     then srincor := false
     else begin
          k := 1;
          repbr;
          if reponse = 'n' then srincor := false
          end
     end
end;

procedure verre;
(*********************************)
(* COHERENCE AU NIVEAU DU RESEAU *)
(*********************************)
begin
mess51;
repinvalide := true;
while repinvalide do
     begin
     inoui(reponse,code);
     case code of
          blank    : vidoui;
          help     : herep8;
          invalid  : invoui;
          ok       : repinvalide := false
          end
     end;
if reponse = 'n'
then rincor := false
else begin
     for i := 1 to bores do reseau[i] := 0;
     for i := 1 to bolnt do decrit[i] := false;
     for i := 1 to bolnt do deb[i] := 0;
     for i := 1 to bolnt do fin[i] := 0;
     adc := 1
     end
end;
```

```
(*****************************************************************************)
(*                                                                           *)
(*                          PROCEDURE D'AFFICHAGE DU RESEAU                  *)
(*                                                                           *)
(*****************************************************************************)

procedure impre;
begin
mess52;
repinvalide := true;
while repinvalide do
     begin
     inoui(reponse,code);
     case code of
          blank   : vidoui;
          help    : heimp;
          invalid : invoui;
          ok      : repinvalide := false
          end
     end;
if reponse = 'o'
then begin
     i := 1;
     while i <= nbnt do
          begin
          pe := -reseau[deb[i]];
          ps := -reseau[fin[i]];
          mess53(tnt[i],pe,ps);
          j := deb[i];
          dernier := false;
          while not dernier do
               begin
               num := -reseau[j];
               k := 1;
               while (reseau[j+k] > 0) do
                    begin
                    if ((j+k) = -reseau[reseau[j+k]-1])
                    then begin
                         l := 0;
                         while reseau[reseau[j+k]+l] >= 0 do l := l + 1;
                         l := -reseau[reseau[j+k]+l];
                         m := 1;
                         while reseau[l+m] > 0 do m := m + 1;
                         proc := -reseau[l+m];
                         mess54(num,k,proc,m);
                         l := 0;
                         while reseau[reseau[j+k]+l] >= 0 do
                              begin
                              mess55(reseau[reseau[j+k]+l]);
                              l := l + 1
                              end;
                         mess56
                         end;
                    k := k + 1
                    end;
               j := j + k;
               if num = ps then dernier := true
```

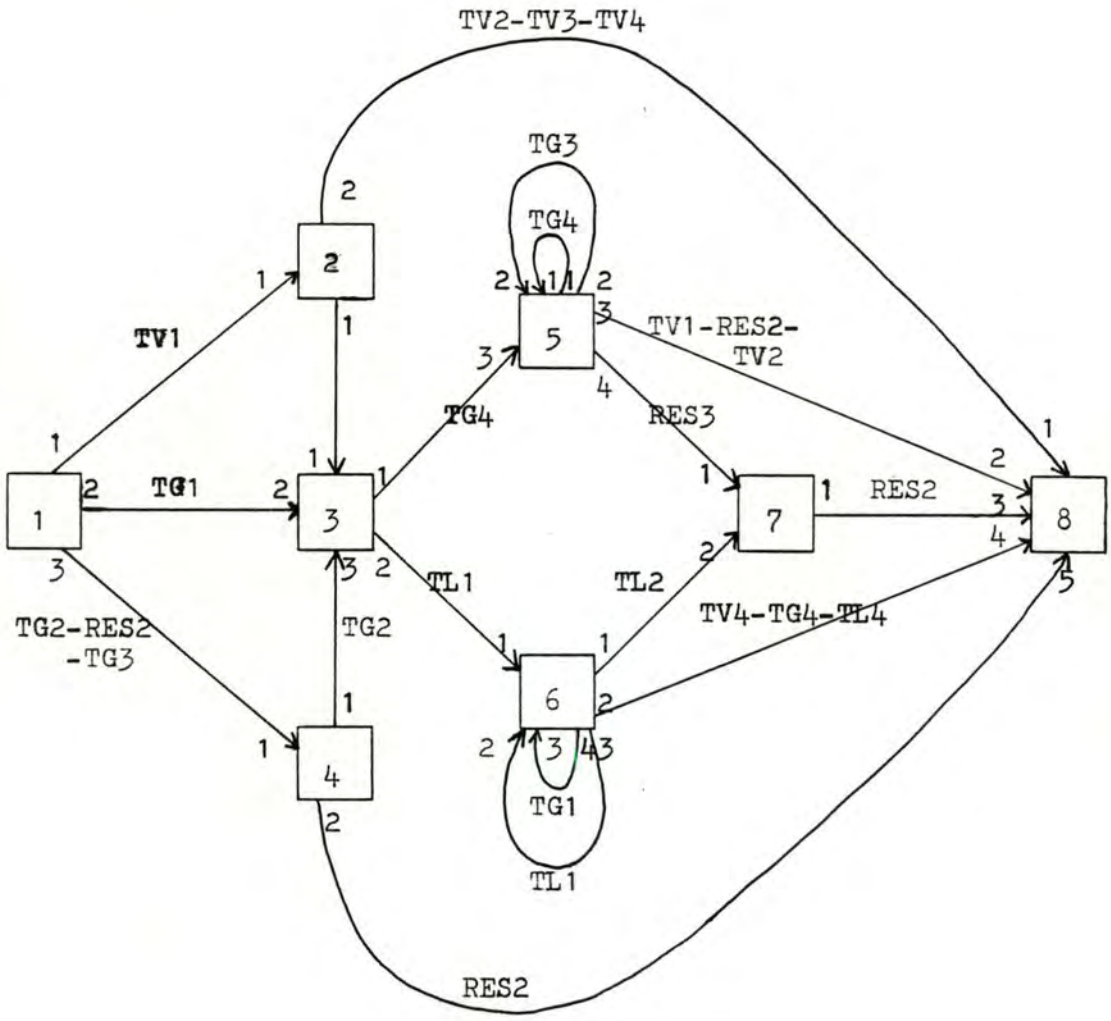
```
                end;
            i := i + 1
            end
      end
end;

(*************************************************************************)
(*                                                                       *)
(*                  PROCEDURE DE MEMORISATION DU RESEAU                  *)
(*                                                                       *)
(*************************************************************************)

procedure ecrre;
begin
rewrite (res,'RNP');
writeln (res,adc);
writeln (res,nbnt);
for i := 1 to bo1nt do for j := 1 to bo2nt do write (res,tnt[i,j]);
writeln (res);
writeln (res,nbtv);
for i := 1 to bo1tv do for j := 1 to bo2tv do write (res,ttv[i,j]);
writeln (res);
writeln (res,nbtl);
for i := 1 to bo1tl do for j := 1 to bo2tl do write (res,ttl[i,j]);
writeln (res);
writeln (res,nbtg);
for i := 1 to bo1tg do for j := 1 to bo2tg do write (res,ttg[i,j]);
writeln (res);
for i := 1 to bo1nt do writeln (res,deb[i]);
for i := 1 to bo1nt do writeln (res,fin[i]);
for i := 1 to bores do writeln (res,reseau[i])
end;

(*************************************************************************)
(*                                                                       *)
(*                          PROGRAMME PRINCIPAL                          *)
(*                          ___________________                          *)
(*                                                                       *)
(*************************************************************************)

(*************************************************************************)
(*                                                                       *)
(*                          PARTIE INITIALISATION                        *)
(*                                                                       *)
(*************************************************************************)

begin
mess0;
initnt;
inittv;
inittl;
inittg;
mess15;
reponse := 'o';
while reponse = 'o' do
      begin
      modini
```

```
     end;
for i := 1 to bores do reseau[i] := 0;
for i := 1 to boint do decrit[i] := false;
for i := 1 to boint do deb[i] := 0;
for i := 1 to boint do fin[i] := 0;
adc := 1;

(*****************************************************************************)
(*                                                                           *)
(*                          PARTIE TRAITEMENT DU RESEAU                      *)
(*                                                                           *)
(*****************************************************************************)

mess24c;
rincor := true;
while rincor do
     begin
     i := 1;
     while (i <= nbnt) do
          begin
          srincor := true;
          nincor := true;
          dsrincor := true;
          while srincor do
               begin
               while nincor do
                    begin
                    while dsrincor do
                         begin
                         trdsr
                         end;
                    mess32b;
                    reprsr := false;
                    j := 1;
                    while (j <= nbp) and (not reprsr) do
                         begin
                         trno
                         end;
                    if not reprsr
                    then verno
                    end;
               deb[ind] := ad(pe,table,nbp);
               fin[ind] := ad(ps,table,nbp);
               mess38e;
               j := 1;
               while (j <= nbp) and (not reprsr) do
                    begin
                    trbr
                    end;
               if not reprsr
               then verbr
               end;
          i := i + 1
          end;
     impre;
     verre
     end;
```

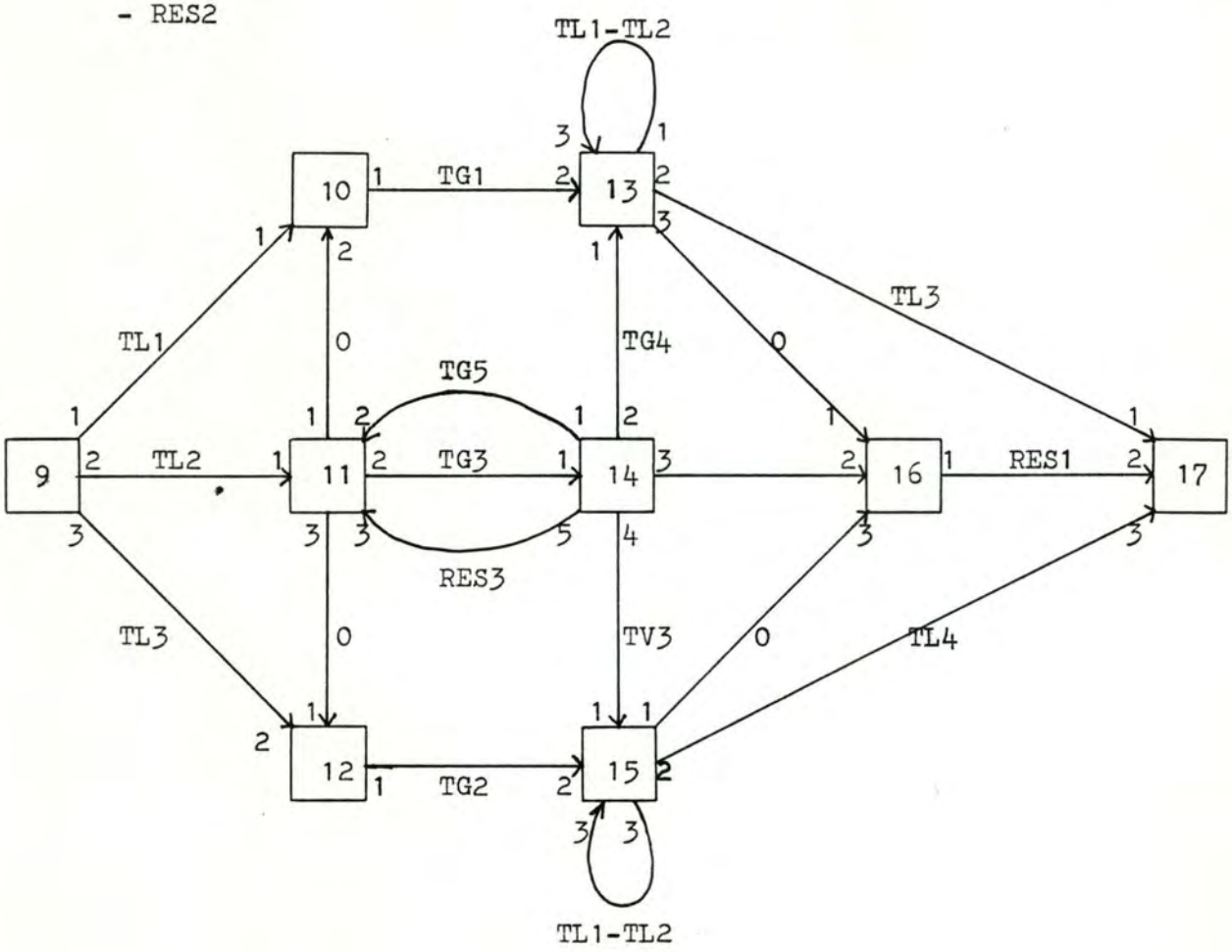
```
(*****************************************************************************)
(*                                                                           *)
(*                          PARTIE AFFICHAGE DU RESEAU                       *)
(*                                                                           *)
(*****************************************************************************)

ecrre
end.
```

Pour donner un exemple de résultats du programme CRERES, nous avons pris un RNP bidon dont la représentation graphique est donnée par les réseaux suivants :

- RES1



- RES2



- RES3

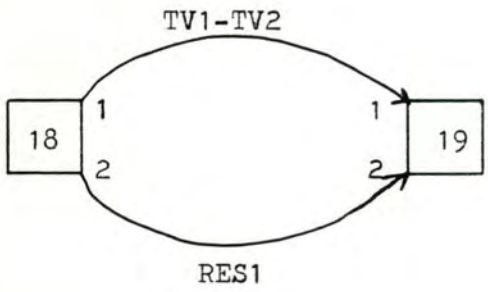


fig. A.II.1. : Exemple de réseau à noeuds procéduraux

Nous avons donc les éléments suivants :

- non-terminaux : RES1, RES2 et RES3;
- terminaux vrais : TV1, TV2, TV3 et TV4;
- terminaux lexiques figés : TL1, TL2, TL3 et TL4;
- terminaux généraux : TG1, TG2, TG3, TG4 et TG5.

Ce qui nous donne les résultats suivants sur fichier après exécution du programme de création :
(voir listing page suivante) où

- 231 ⟶ 1er élément libre dans le tableau contenant la représentation interne du réseau;
- 3 ⟶ nombre de non-terminaux;
- RES1, RES2, RES3 ⟶ ensemble de non-terminaux;
- 4 ⟶ nombre de terminaux vrais;
- TV1, TV2, TV3, TV4 ⟶ ensemble de terminaux vrais;
- 4 ⟶ nombre de terminaux lexiques figés;
- TL1, TL2, TL3, TL4 ⟶ ensemble de terminaux lexiques figés;
- 5 ⟶ nombre de terminaux généraux;
- TG1, TG2, TG3, TG4, TG5 ⟶ ensemble de terminaux généraux;
- -1, 107, 218, 0, 0... ⟶ tableau contenant les pointeurs vers les noeuds procéduraux d'entrée du RNP;
- 44, 155, 223, 0, 0... ⟶ tableau contenant les pointeurs vers les noeuds procéduraux de sortie du RNP;
- -1, 46, 49, 52... ⟶ tableau contenant la représentation interne du RNP.

Rappelons encore que les différents types de terminaux correspondent aux partitions du lexique :

- Lexique indépendant de l'application :
  - = terminaux vrais : correspondent aux mots de liaison apparaissant de manière explicite dans la définition du RNP;
  - = terminaux lexiques figés: correspondent aux sous-classes grammaticales.
- Lexique propre à chaque application :
  - = terminaux généraux : correspondent aux noms, verbes etc.

	71	2001	188	-151
231	74	2	-16	1
3	65	2002	216	-153
res1res2res3	80	-42	209	-18
4	77	-26	216	225
tv1 tv2 tv3 tv4	-5	3	185	229
4	77	-36	-17	229
tl1 tl2 tl3 tl4	80	-31	-108	226
5	83	1002	1001	-19
tg1 tg2 tg3 tg4 tg5	88	-35	-112	-219
1	102	-32	-109	2001
107	99	2004	1002	2002
218	68	3004	-117	-222
0	-6	1004	-110	-220
0	91	-40	1003	1
0	94	-33	-122	-221
0	99	1001	-114	0
0	102	-28	3001	0
0	91	-34	-127	0
0	88	3001	-119	0
0	-7	-27	0	0
0	105	-38	-111	0
0	74	2	-120	0
0	96	-41	0	0
0	105	-9	-123	0
0	85	157	-121	0
0	62	160	3003	0
0	-8	163	-133	0
0	-2	169	-125	0
0	2001	157	3002	0
44	-5	-10	-141	0
155	-3	166	-130	0
223	3001	203	1001	0
0	-10	191	1002	0
0	-4	160	-126	0
0	3002	-11	-131	0
0	2	169	1003	0
0	3003	172	-154	0
0	-15	175	-132	0
0	-7	163	0	0
0	3002	172	-149	0
0	-11	-12	-135	0
0	-8	178	3005	0
0	2002	182	-116	0
0	2003	166	-136	0
0	2004	194	3004	0
0	-43	-13	-128	0
0	-13	181	-137	0
0	3004	185	1003	0
0	-19	188	-148	0
-1	-14	175	-138	0
46	1001	-14	2003	0
49	-29	191	-142	
52	-17	194	-139	
46	3002	197	3	
-2	-9	200	-115	
57	-18	203	-144	
60	2	213	0	
71	-39	178	-147	
49	-23	200	-145	
57	3004	-15	1004	
-3	-21	206	-152	
65	-24	209	-146	
68	3003	212	1001	
54	-20	206	1002	
-4	-25	197	-140	

A N N E X E III

Manuel d'utilisation du programme CRERES

MANUEL D'UTILISATION DU PROGRAMME CRERES

0. TABLE DES MATIERES

0. TABLE DES MATIERES 0-1
1. INTRODUCTION 1-1
2. UTILISATION DU PROGRAMME CRERES 2-1
2.1. DESCRIPTION D'UNE SESSION DE TRAVAIL 2-1
2.2. INITIALISATION DU RESEAU 2-1
2.2.1. Définition des non-terminaux 2-2
2.2.2. Définition des terminaux vrais 2-2
2.2.3. Définition des terminaux lexiques figés 2-3
2.2.4. Définition des terminaux généraux 2-3
2.2.5. Vérification 2-3
2.2.6. Erreurs 2-4
2.2.6.1. Au niveau du nombre 2-4
2.2.6.2. Au niveau des éléments 2-4
2.2.6.3. Au niveau d'un oui ou non 2-4
2.3. TRAITEMENT DU RESEAU 2-5
2.3.1. Initialisation d'un sous-réseau 2-5
2.3.1.1. Le nom du sous-réseau 2-5
2.3.1.2. Numéro de la procédure d'entrée 2-6
2.3.1.3. Numéro de la procédure de sortie 2-6
2.3.1.4. Nombre de noeuds procéduraux 2-6
2.3.1.5. Vérification 2-6
2.3.1.6. Erreurs 2-7
2.3.2. Traitement des noeuds procéduraux 2-8
2.3.2.1. Le numéro de la procédure 2-8
2.3.2.2. Le nombre d'entrées 2-8
2.3.2.3. Le nombre de sorties 2-8
2.3.2.4. Vérification 2-9
2.3.2.5. Erreurs 2-9
2.3.3. Traitement des branches linéaires 2-10
2.3.3.1. Procédure d'arrivée, numéro d'entrée 2-10
2.3.3.2. Procédure d'arrivée, numéro de la procédu 2-10
2.3.3.3. Les arcs composant la branche 2-10
2.3.3.4. Vérification 2-11
2.3.3.5. Erreurs 2-11
2.3.4. Vérification 2-12
2.4. ECRITURE DU RESEAU 2-13

A. ANNEXE A A-1
B. ANNEXE B B-1
C. ANNEXE C C-1

1. INTRODUCTION

- Le programme CRERES est un logiciel qui vous permet de passer interactivement de la représentation graphique d'un réseau à noeuds procéduraux (RNP) à sa représentation interne en machine.

- Au cours du traitement, le programme vérifie au maximum la cohérence des données introduites. Si vous avez commis une erreur, vous avez différentes options de reprise suivant l'étape du traitement où l'erreur a été commise.

- Une étape étant terminée et reconnue comme cohérente, vous avez quand même la possibilité de revenir en arrière et de reprendre le traitement à un stade antérieur.

- Il n'y a pas de formar fixe pour les données à introduire, elles peuvent se trouver n'importe où sur la ligne.

- A chaque demande du programme, vous pouvez répondre par '?'. Vous aurez alors plus de renseignements sur ce qu'il y a à faire. Après l'affichage de ce message 'help', le traitement reprend normalement, càd que vous devez répondre à la demande précédant le message.

- Chaque ligne affichée au terminal est précedée d'un caractère vous indiquant le type de la ligne :

  '!' ----> communication simple
  '?' ----> demande de données
  '#' ----> message d'erreur
  '*' ----> message help

- Quelques conventions de vocabulaire :

réseau	:	c'est l'ensemble que vous avez à décrire avec le programme CRERES. C'est votre modélisation du langage avec l'outil que sont les RNP.
sous-réseau	:	c'est un composant du réseau, formant en lui même une entité logique et cohérente.
procédure	:	ou encore noeud-procédural : un noeud d'un sous-réseau.
branche	:	chemin reliant deux noeuds d'un sous-réseau.
arc	:	composant d'une branche.

- Chaque demande exigeant comme réponse oui ou non accepte :

O,o
OU,ou
OUI,oui
} pour oui

N,n
NO,no
NON,non
} pour non

2. UTILISATION DU PROGRAMME CRERES

2.1. DESCRIPTION D'UNE SESSION DE TRAVAIL

Une session commence par l'appel du programme :

CRERES

La première phase est la phase d'initialisation du réseau. On définit ici les valeurs de base du réseau. Après vérification par l'utilisateur, on passe à la phase de traitement du réseau, qui se décompose en trois sous-étapes, pour chaque sous-réseau à décrire. C'est l'initialisation du sous-réseau, le traitement des noeuds du sous-réseau et le traitement des branches du sous-réseau. Après avoir traité tous les sous-réseaux, on peut faire une dernière vérification sur le réseau et passer ensuite à la phase d'écriture du réseau.

Le résultat du traitement se retrouve dans le fichier RNP.

2.2. INITIALISATION DU RESEAU

Cette phase sert à définir les valeurs de base du réseau qui sont :

- nombre de non-terminaux
- l'ensemble des non-terminaux
- nombre de terminaux vrais
- l'ensemble des terminaux vrais
- nombre de terminaux lexiques figés
- l'ensemble des terminaux lexiques figés
- nombre de terminaux généraux
- l'ensemble des terminaux généraux

2.2.1. Définition des non-terminaux

Après l'affichage d'un message vous informant que vous travaillez avec le programme CRERES, vous devez définir les non-terminaux, càd :

- introduire le nombre de non-terminaux.
  Ce nombre doit être un entier.

- introduire l' ensemble des non-terminaux.
  Vous devez pour cela taper tous les éléments constituant cet ensemble. Ces éléments doivent être séparés par au moins un blanc et se trouver sur une même ligne, càd sans C.R. intermédiaire.

Pour les limitations concernant ces données consulter l'annexe A.

2.2.2. Définition des terminaux vrais

Vous devez ensuite définir vos terminaux vrais, càd :

- introduire le nombre de terminaux vrais.
  Ce nombre doit être un entier.

- introduire l'ensemble des terminaux vrais.
  Vous devez pour cela taper tous les éléments constituant cet ensemble. Ces éléments doivent être séparés par au moins un blanc et se trouver sur une même ligne, càd sans C.R. intermédiaire.

Pour les limitations concernant ces données, voir annexe A.

2.2.3. Définition des terminaux lexiques figés

Même procédé que pour les terminaux vrais.

2.2.4. Définition des terminaux généraux

Même procédé que pour les terminaux vrais.

2.2.5. Vérification

Dans un but de vérification visuelle par l'utilisateur, l'ensemble des non-terminaux, terminaux vrais, terminaux lexiques figés et terminaux généraux est affiché à l'écran. Vous avez ensuite la possibilité de modifier cette définition en répondant oui à la question affichée. De plus, vous avez la possibilité de ne modifier que certains types d'éléments, soit :

- les non-terminaux
- les terminaux vrais
- les terminaux lexiques figés
- les terminaux généraux

Cela se fait de la manière suivante : Après avoir répondu oui à la question de modification, le programme vous demande pour chaque type si vous voulez le modifier. Pour le modifier, il faut répondre oui, sinon il faut répondre non. Après avoir passé en revue tous les types, on revient à l'affichage en vue d'une vérification visuelle.

Si vous avez répondu non à la demande de modification, l'initialisation est considéré comme terminée et vous n'avez plus la possibilité d'y revenir ultérieurement.

2.2.6. Erreurs

2.2.6.1. Au niveau du nombre

Si vous avez commis une erreur au niveau du nombre d'éléments, le programme demande simplement la réintroduction du nombre.

2.2.6.2. Au niveau des éléments

Si vous avez commis une erreur au niveau des éléments, le programme demande la réintroduction des éléments. Dans le cas d'une incohérence entre nombre indiqué et nombre effectif d'éléments, vous pouvez soit recommencer par introduire le nombre en répondant oui à la question posée, soit ne réintroduire que la liste des éléments en répondant non à la question.

2.2.6.3. Au niveau d'un oui ou non

Si vous avez commis une erreur en répondant à une demande d'un oui ou non, le programme demande simplement la réintroduction de la réponse.

2.3. TRAITEMENT DU RESEAU

Le traitement du réseau est en fait le traitement un par un des différents sous-réseaux. Un sous-réseau, une fois décrit, et cette description approuvée, ne peut plus être modifié, à moins de reprendre la description de tous les sous-réseaux.

Le traitement d'un sous-réseau se décompose lui-même en trois phases :

- L'initialisation du sous-réseau
- Le traitement des noeuds procéduraux
- Le traitement des branches linéaires

2.3.1. Initialisation d'un sous-réseau

L'initialisation d'un sous-réseau consiste en la définition des valeurs de base du sous-réseau, càd :

- nom du sous-réseau
- numéro de la procédure d'entrée
- numéro de la procédure de sortie
- nombre de noeuds procéduraux dans le sous-réseau

Pour les limitations sur ces données, voir annaxe B

2.3.1.1. Le nom du sous-réseau

Le nom du sous-réseau doit être un élément figurant parmi les non-terminaux et ne doit pas être un sous-réseau déjà décrit.

2.3.1.2. Numéro de la procédure d'entrée

Le numéro de la procédure d'entrée doit être un entier.

2.3.1.3. Numéro de la procédure de sortie

Le numéro de la procédure de sortie doit être un entier, différent du numéro de la procédure d'entrée.

2.3.1.4. Nombre de noeuds procéduraux

Le nombre de noeuds procéduraux doit être un entier supérieur ou égal à 2. (Comme la procédure d'entrée ne peut être en même temps la procédure de sortie, le nombre minimum de noeuds est 2).

2.3.1.5. Vérification

Après avoir correctement initialisé le sous-réseau, vous avez la possibilité de reprendre cette initialisation si vous voulez modifier une donnée introduite pendant cette intialisation. Pour ce faire, répondez oui à la demande de modification. Si vous voulez continuer le traitement du sous-réseau, répondez non.

2.3.1.6. Erreurs

Des erreurs au niveau de l'initialisation du sous-réseau n'entraînent qu'une demande de réintroduction de la donnée par le programme.

2.3.2. Traitement des noeuds procéduraux

Après l'initialisation du sous-réseau, on arrive à la description des noeuds procéduraux. Pour chaque procédure, il faut alors introduire :

- le numéro de la procédure
- le nombre d'entrées
- le nombre de sorties

2.3.2.1. Le numéro de la procédure

Le numéro de la procédure doit être un entier. La première procédure décrite doit obligatoirement être la procédure d'entrée tandis que la dernière procédure décrite doit être la procédure de sortie.

2.3.2.2. Le nombre d'entrées

Le nombre d'entrées doit être un entier. *)

2.3.2.3. Le nombre de sorties

Le nombre de sorties doit être un entier. *)

*) Les procédures qui ne sont ni procédures d'entrée ni procédures de sortie, doivent avoir plus de 0 entrées et sorties.

2.3.2.4. Vérification

- Si après avoir décrit toutes les procédures, vos données sont incohérentes, plus d'entrées que de sorties dans un sous-réseau, vous avez deux options de reprise :

  * à l'initialisation du sous-réseau
  * au début du traitement des noeuds

  Si vous voulez reprendre l'initialisation du sous-réseau répondez oui à la demande de reprise de l'initialisation et reprenez alors au niveau de l'initialisation.
  Si vous ne désirez pas réinitialiser le sous-réseau, répondez non. Vous devez alors reprendre au niveau de la description des noeuds et répondre oui à la question de reprise des noeuds. Si vous répondez nom, le sous-réseau est toujours dans un état incohérent et vous avez de nouveau les deux options.

- Si par contre, vos données sont cohérentes, vous avez quand même la possibilité d'une reprise. Les options sont les mêmes que dans le cas d'une description erronnée. Pour reprendre, répondez oui à la demande de modification. Pour reprendre au niveau initialisation, répondez oui à la demande de reprise au niveau intialisation, sinon répondez non. Pour reprendre au niveau traitement des noeuds, répondez oui à la demande de reprise au niveau des noeuds. Si vous répondez non, votre description étant cohérente, vous passez à la phase de traitement des branches linéaires.

- Si votre description est cohérente et que vous ne voulez pas modifier de données, répondez non à la demande de modification. Vous passez alors dans la phase traitement des branches liméaires.

2.3.2.5. Erreurs

Si en cours de description des noeuds, vous introduisez une donnée incohérente, les mêmes options de reprise que précédemment s'offrent. Si vous ne désirez ni reprendre à l'initialisation, ni au début de la description des noeuds, vous devez réintroduire la description erronnée.

2.3.3. Traitement des branches linéaires

Dans cette phase, il faut décrire les branches du sous-réseau, càd, pour une branche, sortant d'une procédure donnée par une sortie donnée, indiquer :

- le numéro de la procédure d'arrivée de la branche
- le numéro de l'entrée dans la procédure
- les arcs composant les branches

Pour les limitations concernant ces données, voir annexe C.

2.3.3.1. Procédure d'arrivée, numéro d'entrée

Le numéro d'entrée doit être un entier. Il ne doit pas être supérieur au nombre d'entrées déclarées pour cette procédure et ne doit pas être une entrée déjà décrite de cette procédure.

2.3.3.2. Procédure d'arrivée, numéro de la procédure

Le numéro de la procédure doit être un entier. Cette procédure doit déjà avoir été décrite au cours du traitement des noeuds.

2.3.3.3. Les arcs composant la branche

Les arcs doivent être des éléments décrits soit dans l'ensemble des non-terminaux, soit dans l'ensemble des terminaux généraux, de type lexiques figés ou vrais. Les éléments doivent être séparés par des blancs, un au moins.

2.3.3.4. Vérification

Après avoir décrit toutes les branches :

- Si votre description est incohérente, vous avez les options de reprise suivantes :

  - à l'initialisation du sous-réseau
  - au début du traitement des noeuds
  - au début du traitement des branches

  Si vous n'avez pris aucune option de reprise, votre description étant toujours incohérente, vous avez de nouveau les trois options de reprise.

- Si votre description est cohérente, vous pouvez modifier le sous-réseau en répondant oui à la demande de modification et en prenant une des trois options de reprise. Si vous n'en prenez aucune, votre description étant cohérente, vous passez au traitement du sous-réseau suivant.

- Si votre description est cohérente et que vous ne voulez rien modifier, répondez non à la demande de modification. Vous passez alors à l'étape suivante, càd le traitement du sous-réseau suivant.

2.3.3.5. Erreurs

Si en cours de description des branches, vous avez commis une erreur, vous avez également les trois options de reprise. Si vous n'en prenez aucune, il faut réintroduire la description de la branche erronnée.

2.3.4. Vérification

Si tous les sous-réseau sont traités d'une manière cohérente, vous pouvez afficher à l'écran la description de tout le réseau en répondant oui à la demande d'affichage. Si vous ne désirez pas cet affichage, répondez non.

Ensuite vous avez la possibilité de reprendre la description du réseau, càd à partir de la description du premier sous-réseau. Vous oubliez donc tout ce que vous avez fait à propos des sous-réseaux. Pour reprendre répondez oui, sinon répondez non.

2.4. ECRITURE DU RESEAU

Dans cette phase, le réseau et les informations nécessaires à son interprétation sont écrites sur le fichier RNP. L'utilisateur n'a plus rien à faire que d'attendre la fin de l'exécution du programme.

(version du 8/11/82)

ANNEXE A

Limitations des valeurs de base du réseau :

- nombre minimum d'éléments par type : 1
- " maximum de non-terminaux : 20
- " " " terminaux vrais : 20
- " " " terminaux lexiques figés : 30
- " " " terminaux généraux : 20

nombre maximum de caractères pris en compte :

- pour les non-terminaux : les 4 premiers
- " " terminaux vrais : " " "
- " " terminaux lexiques figés : " " "
- " " terminaux généraux : " " "

(version du 8/11/82)

ANNEXE C

Limitations du sous-réseau, branches :

- nombre maximum d'arcs par branche : 10
- nombre maximum de caractères pris en compte par arc :

  les 4 premiers

A N N E X E IV

Dossier d'analyse de l'analyseur syntaxique

DOSSIER D'ANALYSE DU PROGRAMME

ANSYNT

0. TABLE DES MATIERES

0. TABLE DES MATIERES 0-1
1. SPECIFICATION DU PROBLEME 1-1
2. ARCHITECTURE GENERALE DU SYSTEME 2-1
   2.1. DECOMPOSITION EN MODULES 2-1
   2.2. JUSTIFICATION 2-2
      2.2.1. Séparation ANALYSE-LECTURE 2-2
      2.2.2. Séparation PARCOURS-REPRISE 2-2
      2.2.3. Les modules AFFICHAGE-ARBRE
             GESTION-REPRISES
             BRANCHES-LINEAIRES
             NOEUDS-PROCEDURAUX 2-2
3. SPECIFICATION DES MODULES 3-1
   3.1. ANALYSEUR 3-1
      3.1.1. Spécification des traitements 3-1
      3.1.2. Description des messages 3-1
   3.2. LECTURE-RESEAU 3-2
      3.2.1. Spécification des traitements 3-2
      3.2.2. Description des données 3-2
      3.2.3. Description des messages 3-2
   3.3. ANALYSE-PHRASES 3-3
      3.3.1. Spécification des traitements 3-3
      3.3.2. Description des données 3-3
      3.3.3. Description des messages 3-4
   3.4. PARCOURS-SOUS-RESEAU 3-5
      3.4.1. Spécification des traitements 3-5
      3.4.2. Description des données 3-5
      3.4.3. Description des messages 3-6
   3.5. REPRISE-SOUS-RESEAU 3-7
      3.5.1. Spécification des traitements 3-7
      3.5.2. Description des données 3-7
      3.5.3. Description des messages 3-8
   3.6. NOEUDS-PROCEDURAUX 3-9
      3.6.1. Spécification des traitements 3-9
      3.6.2. Description des données 3-9
      3.6.3. Description des messages 3-9
   3.7. PROCEDURES ASSOCIEES AUX NOEUDS 3-10
      3.7.1. Spécification des traitements 3-10
      3.7.2. Description des données 3-10
   3.8. BRANCHES-LINEAIRES 3-11
      3.8.1. Spécification des traitements 3-11
      3.8.2. Description des données 3-11
      3.8.3. Description des messages 3-11
   3.9. PROCEDURES ASSOCIEES AUX BRANCHES 3-12
      3.9.1. Spécification des traitements 3-12
      3.9.2. Description des données 3-12
      3.9.3. Description des messages 3-12

3.10. GESTION-REPRISES 3-13
3.10.1. Spécification des traitements 3-13
3.10.2. Description des données 3-13
3.10.3. Description des messages 3-13
3.11. AFFICHAGE-ARBRE 3-14
3.11.1. Spécication des traitements 3-14
3.11.2. Description des données 3-14
3.11.3. Description des messages 3-14
4. LISTINGS 4-1

1. SPECIFICATION DU PROBLEME

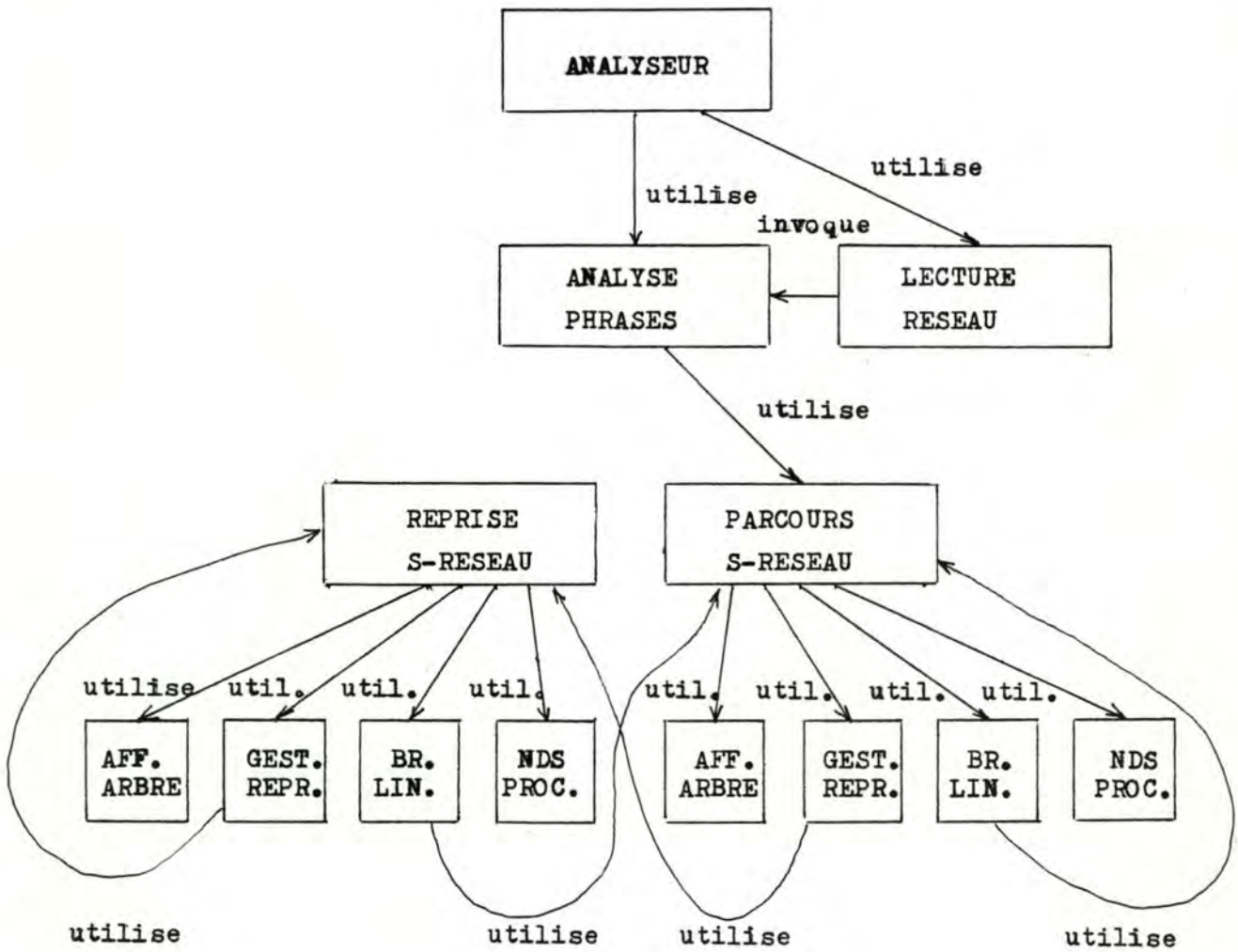
Construire un analyseur syntaxique, qui a partir d'un lexique et d'un niveau phonétique entièrement simulés, et d'un RNP sous représentation interne, analyse une phrase en français.

Lexique et niveau phonétique simulé signifie que tous les accès au lexique, au treilli de phonèmes ou au signal acoustique sont simulés. On n'accède donc pas réellement au lexique ou au signal, mais c'est d'une manière interactive, l'utilisateur répondant à un certain nombre de questions, que l'analyseur reçoit les informations dont il a besoin pour travailler. Dans le cas de reconnaissances erronnées, il doit pouvoir être capable d'effectuer des retours en arrière.

L'analyseur est conçu pour une analyse gauche-droite, utilisant une stratégie du meilleur d'abord.

2. ARCHITECTURE GENERALE DU SYSTEME

2.1. DECOMPOSITION EN MODULES



2.2. JUSTIFICATION

2.2.1. Séparation ANALYSE-LECTURE

Les modules ANALYSE-PHRASES et LECTURE-RESEAU ont été séparés pour la raison évidente qu'ils n'ont rien à faire l'un avec l'autre. En effet, LECTURE-RESEAU sera responsable de l'acquisition de la représentation interne du réseau à noeuds procéduraux, tandis que ANALYSE-PHRASES et ses sous-modules seront chargés de le parcourir, une fois ce dernier acquis.

2.2.2. Séparation PARCOURS-REPRISE

Les modules PARCOURS et REPRISE ont été séparés parcequ'ils représentent conceptuellement deux actions différentes. Dans PARCOURS, on va en effet parcourir de la gauche vers la droite un sous-réseau, qui, dans ce contexte, n'a pas encore été parcouru. Dans REPRISE cependant, on va reprendre l'analyse d'un sous-réseau qu'on a déjà entièrement parcouru, ceci pour permettre des retours en arrière. On peut quand même constater dès maintenant que ces modules ne vont pas différer fondamentalement et qu'ils vont utiliser les mêmes sous-modules.

2.2.3. Les modules
- AFFICHAGE-ARBRE
- GESTION-REPRISES
- BRANCHES-LINEAIRES
- NOEUDS-PROCEDURAUX

AFFICHAGE-ARBRE a été séparé du reste, parceque c'est un module qui va disparaître du système une fois celui-ci complet, càd quand les différents accès au lexique, au niveau acoustique etc ne seront plus simulés et lorsqu'on ne désire plus avoir d'arbre syntaxique à l'écran pour des raisons de vérification.

Les autres modules ont été séparés parcequ'ils effectuent des actions très spécifiques et indépendantes. GESTION-REPRISES s'occupe de gérer les retours en arrière, BRANCHES-LINEAIRES traite les branches qu'on retrouve lors du parcours, tandis que NOEUDS-PROCEDURAUX s'occupe des noeuds.

Nous allons ici spécifier les modules du programme ANSYNT dans la version à simple indéterminisme, ce qui signifie qu'on ne peut

3.3. ANALYSE-PHRASES

3.3.1. Spécification des traitements

Le module ANALYSE-PHRASES demande à l'utilisateur s'il a une phrase à analyser par un message de demande de phrase à analyser. L'utilisateur répond par un message phrase à analyser.

1. Si la réponse est positive, il demande à l'utilisateur quel est le premier sous-réseau à parcourir par un message de demande de premier sous-réseau à parcourir. L'utilisateur répond par un message premier sous-réseau à parcourir. Le module doit vérifier ensuite qu'il s'agit bien d'un sous-réseau ayant été décrit.

   1.1. S'il ne s'agit pas d'un sous-réseau, on en avertit l'utilisateur par un message d'erreur.

   1.2. S'il s'agit d'un sous-réseau, il crée et initialise une pile et lance le module PARCOURS avec le message premier sous-réseau à parcourir et le message sous-réseau appelant (qui est 0 car on n'a pas encore de sous-réseau appelant). Le module ANALYSE-PHRASES en reçoit la réponse message résultat parcours. En fonction de ce résultat il envoie un message résultat analyse à l'utilisateur.

   Ensuite il redemande alors à l'utilisateur s'il a encore une phrase à analyser par un message de demande de phrase à analyser. L'utilisateur répond par un message phrase à analyser et le module reprend le traitement au début.
2. Si la réponse est négative, le module se termine.

3.3.2. Description des données

Le module ANALYSE-PHRASES travaille sur les mêmes données que le module LECTURE-RESEAU, à l'exception de la donnée REPONSE (nom du fichier), qui n'est pas accessible à ANALYSE-PHRASES. De plus, le module utilise une pile dont les éléments ont la structure :

| P1 | S | E | P2 |

avec comme signification pour les différents composants :

- P1 --- numéro de la procédure de laquelle on sort
- S --- numéro de la sortie par laquelle on sort de la procédure
- E --- numéro de l'entrée par laquelle on entre dans la procédure
- P2 --- numéro de la procédure dans laquelle on entre

Ces données sont globales et accessibles à tous les sous-modules.

3.3.3. Description des messages

Messages externes :	d'entrée :	message phrase à analyser
		message premier sous-réseau à parcourir
	de sortie:	message de demande de phrase à analyser
		message de demande de premier sous-réseau à parcourir
		message d'erreur
		message résultat analyse
internes :	d'entrée :	message résultat parcours
	de sortie:	message premier sous-réseau à parcourir
		message sous-réseau appelant

3.4. PARCOURS-SOUS-RESEAU

3.4.1. Spécification des traitements

Il parcourt le sous-réseau déterminé par message sous-réseau à parcourir, n'ayant pas encore été analysé dans ce contexte, en partant de la gauche, càd de la procédure d'entrée. Pour l'explorer, il emprunte un chemin passant par un certain nombre de noeuds et d'arcs. Chaque fois qu'il se trouve sur un noeud, ce qui est indiqué par position, il active le module NOEUDS-PROCEDURAUX avec message procédure à activer, message numéro entrée et message sous-réseau appelant, tandis que s'il se trouve sur un arc, il active le module BRANCHES-LINEAIRES avec message branche à parcourir et message sous-réseau parcouru. Le résultat du parcours d'un arc lui est signalé par message résultat parcours. Si le parcours ne s'est pas bien passé, on active GESTION-REPRISES avec message sous-réseau parcouru et message position. GESTION-REPRISES donne comme réponse message position et message résultat reprise. Si la reprise s'est bien passée, message position indique le prochain arc à parcourir, sinon c'est une valeur indéterminée. Dans tous les cas, sauf quand la reprise n'a pas réussie, on active AFFICHAGE-ARBRE avec message position. Pour savoir quel arc parcourir après un noeud, le module doit consulter le sommet de la pile qui lui indique le numéro le numéro de la sortie du noeud. De cette manière, ou bien le module trouve un chemin jusqu'à la procédure de sortie du sous-réseau ou bien il n'en trouve pas. Ce résultat est communiqué au module appelant par message résultat parcours.

3.4.2. Description des données

Le module PARCOURS-SOUS-RESEAU utilise les mêmes données que le module ANALYSE-PHRASES avec, en plus une donnée :

- POSITION --- indiquant la position dans le sous-réseau

3.4.3. Description des messages

Messages externes :	d'entrée :	pas de message
	de sortie:	pas de message
internes :	d'entrée :	message sous-réseau à parcourir
		message sous-réseau appelant
		message résultat parcours
		message résultat reprise
		message position
	de sortie:	message résultat parcours
		message procédure à activer
		message numéro entrée
		message sous-réseau appelant
		message branche à parcourir
		message sous-réseau parcouru
		message position

3.5. REPRISE-SOUS-RESEAU

3.5.1. Spécification des traitements

Le module REPRISE-SOUS-RESEAU reprend un sous-réseau, déterminé par message sous-réseau à reprendre et ayant déjà été analysé dans ce contexte, en partant de la droite, càd de la procédure de sortie du sous-réseau, et en essayant de retrouver un point de reprise dans ce sous-réseau. Ce point de reprise trouvé, les traitements seront les mêmes que dans PARCOURS-SOUS-RESEAU, càd recherche d'un nouveau chemin à partir du point de reprise vers la procédure de sortie. Pour cela, il faut de nouveau passer par un certain nombre de noeuds et d'arcs. Chaque fois que le module rencontre un noeud, il active NOEUDS-PROCEDURAUX avec message procédure à activer, message numéro entrée et message sous-réseau appelant, tandis que s'il rencontre un arc, il active BRANCHES-LINEAIRES avec message branche à parcourir et message sous-réseau parcouru. Le résultat du parcours d'un arc lui est signalé par message résultat parcours. Si le parcours ne s'est pas bien passé, on active GESTION-REPRISES avec message position et message sous-réseau parcouru. GESTION-REPRISES donne comme résultat message résultat reprise et message position. Si la reprise s'est bien passée, position indique le prochain arc à parcourir, sinon c'est une valeur indéterminée. Dans tous les cas, sauf quand la reprise n'a pas réussie, on active AFFICHAGE-ARBRE avec message position. Pour savoir quel arc parcourir après un noeud, le module doit consulter le sommet de la pile qui lui indique le numéro de sortie du noeud. De cette manière, ou bien le module trouve un chemin jusqu'à la procédure de sortie ou bien il n'en trouve pas. Ce résultat est communiqué au module appelant par message résultat reprise.

3.5.2. Description des données

Le module REPRISES-SOUS-RESEAU utilise les mêmes données que le module ANALYSE-PHRASES avec, en plus une donnée :

- POSITION --- indiquant la position dans le sous-réseau

3.5.3. Description des messages

Messages externes :	d'entrée :	pas de message
	de sortie:	pas de message
internes :	d'entrée :	message sous-réseau à reprendre
		message sous-réseau appelant
		message résultat parcours
		message résultat reprise
		message position
	de sortie:	message résultat parcours
		message procédure à activer
		message numéro entrée
		message sous-réseau appelant
		message branche à parcourir
		message sous-réseau parcouru
		message position

3.6. NOEUDS-PROCEDURAUX

3.6.1. Spécification des traitements

Le module NOEUDS-PROCEDURAUX complète l'élément sommet de la pile avec message numéro d'entrée et message procédure à activer. La procédure correspondant à message procédure à activer est ensuite activée.

3.6.2. Description des données

Mêmes données que le module ANALYSE-PHRASES.

3.6.3. Description des messages

Messages externes :	d'entrée :	pas de message
	de sortie:	pas de message
internes :	d'entrée :	message procédure à activer
		message numéro entrée
		message sous-réseau appelant
	de sortie:	pas de message

3.7. PROCEDURES ASSOCIEES AUX NOEUDS

Nous ne présenterons pas d'analyse détaillée de ces procédures mais uniquement une description sommaire des actions que chaque procédure aura à effectuer.

3.7.1. Spécification des traitements

Les procédures associées aux noeuds doivent prendre en compte des informations de type contextuel, acoustique ou même physique, et, en fonction de ces informations, classer les sorties possibles des noeuds par ordre de préférence sur la pile, la sortie la plus probable se trouvant au sommet de la pile. Dans le cas d'une procédure de sortie d'un sous-réseau, le numéro de la sortie vaut 0. Les éléments inconnus (càd E et P2) sont mis à 0 pour chaque sortie possible.
Dans notre cas, les sorties sont encore simplement empilées par ordre de numérotation.

3.7.2. Description des données

Mêmes données que le module ANALYSE-PHRASES.

3.8. BRANCHES-LINEAIRES

3.8.1. Spécification des traitements

Trois cas peuvent se présenter lors de la réception du message branche à parcourir :

- la branche est vide : on ne fait rien et message résultat parcours est OK
- la branche n'est pas vide et n'appartient pas à l'ensrmble des non-terminaux : on active la procédure associée qui répond par message résultat parcours
- la branche est un NT: on active PARCOURS-RESEAU avec message sous-réseau à parcourir et message sous-réseau appelant qui répond par message résultat parcours.

3.8.2. Description des données

Mêmes données que le module ANALYSE-PHRASES.

3.8.3. Description des messages

Messages externes : d'entrée : pas de message
de sortie: pas de message
internes : d'entrée : message branche à parcourir
message résultat parcours
de sortie: message réaultat parcours

3.9. PROCEDURES ASSOCIEES AUX BRANCHES

3.9.1. Spécification des traitements

Les procédures associées aux branches doivent vérifier si on a bien choisi la bonne branche. On teste donc les hypothèses émises. Elles doivent signaler le résultat au module appelant par message résultat parcours.

Dans notre cas, cette vérification se fait interactivement.

3.9.2. Description des données

Mêmes données que le module ANALYSE-PHRASES.

3.9.3. Description des messages

Messages externes : d'entrée : message vérification hypothèse
de sortie: message demande vérification hypothèse
internes : d'entrée : message branche à parcourir
de sortie: message résultat parcours

3.10. GESTION-REPRISES

3.10.1. Spécification des traitements

Lorsqu'une hypothèse a été rejetée, le module GESTION-REPRISES doit intervenir de manière à ce qu'on puisse reprendre le traitement à un moment antérieur. Il doit modifier l'environnement, càd la pile et la donnée position, de telle sorte que le traitement puisse reprendre au dernier noeud rencontré, qui permette d'essayer une possibilité non encore testée. Le module GESTION-REPRISES ne peut être appelé qu'après un parcours d'arc. Revenir en arrière ne pose pas de problème pour les noeuds ou les arcs qui ne sont pas des non-terminaux. Par contre, s'il faut parcourir en sens inverse un arc qui est un non-terminal, la reprise se fait au niveau de ce sous-réseau. On active alors le module REPRISE-SOUS-RESEAU. Celui-ci donne comme réponse : message résultat reprise. Si ce message est OK, on peut continuer le parcours normal au niveau du sous-réseau ayant activé GESTION-REPRISE. Sinon, il faut continuer à revenir en arrière au niveau de ce sous-réseau. S'il n'y a pas moyen de revenir encore plus loin en arrière, parcequ'on est arrivé à la procédure d'entrée dans le sous-réseau, on le signale par message résultat reprise au module appelant.

3.10.2. Description des données

Mêmes données que le module ANALYSE-PHRASES.

3.10.3. Description des messages

Messages externes : d'entrée : pas de message
de sortie: pas de message
internes : d'entrée : message sous-réseau parcouru
message position
message résultat reprise
de sortie: message résultat reprise
message sous-réseau à reprendre
message sous-réseau parcouru
message position

3.11. AFFICHAGE-ARBRE

3.11.1. Spécification des traitements

Le module AFFICHAGE-ARBRE doit parcourir parallèlement la pile et le réseau et afficher à l'écran l'arbre syntaxique de la phrase reconnue au moment de l'appel du module.

3.11.2. Description des données

Mêmes données que le module ANALYSE-PHRASES.

3.11.3. Description des messages

Messages externes : d'entrée : pas de message
de sortie: message arbre syntaxique
internes : d'entrée : message position
de sortie: pas de message

4. LISTINGS

Suivent les listings des quatre versions dans lesquelles existe l'analyseur syntaxique.

- ANSYNT1 : Cette version est celle décrite dans les pages précédentes. Ses caractéristiques sont :

  + Simple indéterminisme
  + Une unité de compilation

- ANSYNT2 : Cette version diffère de la précédente en ce qu'on a séparé les procédures associées aux noeuds du reste de l'analyseur, celles-ci dépendant du réseau qu'on a décrit. Ses caractéristiques sont :

  + Simple indéterminisme
  + Deux unités de compilation

- ANSYNT3 : Cette version diffère de la première en ce qu'on a introduit un indéterminisme supplémentaire sur les terminaux. Dès lors, plusieurs terminaux peuvent être reconnus pour une même branche avec des csores différents. Il a donc été nécessaire d'introduire une seconde pile pour gérer cet indétermineisme supplémentaire. La structure de l'algorithme cependant n'a pas changé fondamentalement. Ses caractéristiques sont :

  + Double indéterminisme
  + Une unité de compilation

- ANSYNT4 : Cette version diffère de la précédente en ce qu'on a séparé les procédures associées aux noeuds du reste de l'analyseur. Ses caractéristiques sont :

  + Double indéterminisme
  + Deux unités de compilation

A N N E X E V

Listing de l'analyseur

(4 versions sont présentées)

```
(*****************************************************************************)
(*                                                                           *)
(*                          ANALYSEUR-SYNTAXIQUE                             *)
(*                          ====================                             *)
(*                                                                           *)
(*   Cette version de l'analyseur syntaxique comprend la definition des      *)
(*   noeuds-proceduraux dans la definition des procedures de l'analyseur     *)
(*   meme. Si donc on change de reseau, il faut modifier la procedure        *)
(*   'ndsproc' de l'analyseur. Pour eviter cet ennui, il existe une deu-     *)
(*   xieme version de cet analyseur, ou le module noeuds-proceduraux est     *)
(*   declare externe. Cette version peut etre trouvee sur le fichier :       *)
(*                                                                           *)
(*                          ansynt2                                          *)
(*                                                                           *)
(*****************************************************************************)
(*                                                                           *)
(* AUTEUR                       : MOUSEL Pierre                              *)
(* DATE DERNIERE MODIFICATION   : 13/12/82                                   *)
(*                                                                           *)
(*****************************************************************************)

program ansynt(input,output);
(*$h=20000*)

(*****************************************************************************)
(*                                                                           *)
(*              DECLARATION DES DONNEES GLOBALES                             *)
(*              --------------------------------                             *)
(*                                                                           *)
(*****************************************************************************)

const bo1nt = 20;
      bo2nt = 4;
      bo1tv = 20;
      bo2tv = 4;
      bo1tl = 30;
      bo2tl = 4;
      bo1tg = 20;
      bo2tg = 4;
      bores = 2000;
type  elnt     = array [1..bo2nt] of char;
      eltv     = array [1..bo2tv] of char;
      eltl     = array [1..bo2tl] of char;
      eltg     = array [1..bo2tg] of char;
      chaine   = string [bo2nt];
      pointeur = ^elt;
      elt      = record
                   suivant   : pointeur;
                   ps        : integer;
                   s         : integer;
                   e         : integer;
                   pe        : integer;
                   precedent : pointeur
                   end;
var   adc      : integer;
      nbnt     : integer;
      nbtv     : integer;
      nbtl     : integer;
      nbtg     : integer;
      res      : integer;
      sres     : chaine;
      tnt      : array [1..bo1nt] of elnt;
      ttv      : array [1..bo1tv] of eltv;
      ttl      : array [1..bo1tl] of eltl;
      ttg      : array [1..bo1tg] of eltg;
```

```
      deb     : array [1..boint] of integer;
      fin     : array [1..boint] of integer;
      reseau  : array [1..bores] of integer;
      p       : pointeur;
      premier : pointeur;
      dernier : pointeur;

(*************************************************************************)
(*                                                                       *)
(*                          LISTE DES RPOCEDURES                         *)
(*                          --------------------                         *)
(*                                                                       *)
(*************************************************************************)

procedure analyse ; forward;
procedure lecres  ; forward;
procedure anaphr  ; forward;
procedure parcours (    res       : integer;
                    var resultat  : boolean;
                        resapp    : integer); forward;
procedure reprise  (    res       : integer;
                    var resultat  : boolean;
                        resapp    : integer); forward;
procedure modnds   (    finparc   : boolean;
                    var resultat  : boolean;
                        res       : integer;
                        position  : integer;
                        e,pe      : integer;
                        resapp    : integer); forward;
procedure modbr    (var finbr     : boolean;
                    var finparc   : boolean;
                    var resultat  : boolean;
                    var position  : integer;
                    var e,pe      : integer;
                        res       : integer); forward;
procedure ndsproc  (pe : integer ; e : integer ; resapp : integer); forward;
procedure brlin    (    arc       : integer;
                    var resultat  : boolean;
                        res       : integer); forward;
procedure gestrep  (    res       : integer ;
                    var position  : integer ;
                    var resultat  : boolean); forward;
procedure arbre    (position : integer); forward;

(*************************************************************************)
(*                                                                       *)
(*                          LES FONCTIONS UTILISEES                      *)
(*                          -----------------------                      *)
(*                                                                       *)
(*************************************************************************)

function  rechent (position : integer) : integer;
var i : integer;
begin
i := 0;
while (reseau[position + i]) > 0 do i := i + 1;
rechent := i
end;

function rechnd (position : integer) : integer;
begin
while reseau[position] > 0 do position := position - 1;
rechnd := position
end;

(*************************************************************************)
```

```
(*                                                                          *)
(*                          MODULES DE NIVEAU 1                             *)
(*                          -------------------                             *)
(*                                                                          *)
(****************************************************************************)
(****************************************************************************)
(*                                                                          *)
(*                          MODULE ANALYSEUR                                *)
(*                                                                          *)
(****************************************************************************)

procedure analyse;
begin
lecres;
anaphr
end;

(****************************************************************************)
(*                                                                          *)
(*                          MODULES DE NIVEAU 2                             *)
(*                          -------------------                             *)
(*                                                                          *)
(****************************************************************************)
(****************************************************************************)
(*                                                                          *)
(*                          MODULE LECTURE-RESEAU                           *)
(*                                                                          *)
(****************************************************************************)

procedure lecres;
var reponse : string[8];
    res     : text;
    i,j     : integer;
begin
writeln ('* VEUILLEZ ENTRER LE NOM DU FICHIER CONTENANT LE RNP.');
writeln ('* (AU MAXIMUM 8 CARACTERES.)');
readln (reponse);
reset (res,reponse);
readln (res,adc);
readln (res,nbnt);
for i := 1 to bo1nt do for j := 1 to bo2nt do read (res,tnt[i,j]);
readln (res);
readln (res,nbtv);
for i := 1 to bo1tv do for j := 1 to bo2tv do read (res,ttv[i,j]);
readln (res);
readln (res,nbtl);
for i := 1 to bo1tl do for j := 1 to bo2tl do read (res,ttl[i,j]);
readln (res);
readln (res,nbtg);
for i := 1 to bo1tg do for j := 1 to bo2tg do read (res,ttg[i,j]);
readln (res);
for i := 1 to bo1nt do readln (res,deb[i]);
for i := 1 to bo1nt do readln (res,fin[i]);
for i := 1 to bores do readln (res,reseau[i])
end;

(****************************************************************************)
(*                                                                          *)
(*                          MODULE ANALYSE-PHRASES                          *)
(*                                                                          *)
(****************************************************************************)

procedure anaphr;
var  reponse  : char;
     resultat : boolean;
     i        : integer;
```

```
function indice (sres : chaine) : integer;
var i,j    : integer;
    trouve : boolean;
begin
indice := 0;
for i := 1 to nbnt do
     begin
     trouve := true;
     for j := 1 to bo2nt do if sres[j] <> tnt [i,j] then trouve := false;
     if trouve then indice := i
     end
end;

begin
writeln ('* AVEZ VOUS UNE PHRASE A ANALYSER ?');
readln (reponse);
while (reponse = 'o') or (reponse = 'O') do
     begin
     writeln ('* VOUS AVEZ DECIDE D''ANALYSER UNE PHRASE.');
     writeln ('* QUEL EST LE PREMIER SOUS-RESEAU A PARCOURIR ?');
     for i := 1 to bo2nt do sres[i] := ' ';
     readln (sres);
     res := indice (sres);
     if res = 0
     then writeln ('* CE N''EST PAS UN NON-TERMINAL.')
     else begin
          new (p);
          premier := p;
          dernier := p;
          p^.suivant := nil;
          p^.precedent := nil;
          p^.ps := 0;
          p^.s := 0;
          p^.e := 0;
          p^.pe := 0;
          parcours (res,resultat,0);
          if resultat
          then begin
               writeln ('* LA PHRASE A ETE RECONNUE.');
               writeln ('* VOICI L''ARBRE SYNTXIQUE :');
               writeln;
               arbre(adc);
               writeln
               end
          else writeln ('* LA PHRASE N''A PAS ETE RECONNUE.')
          end;
     p := dernier;
     while p <> nil do
          begin
          dernier := dernier^.precedent;
          dispose (p);
          p := dernier
          end;
     writeln ('* AVEZ VOUS ENCORE UNE PHRASE A ANALYSER ?');
     readln (reponse)
     end
end;

(*************************************************************************)
(*                                                                       *)
(*                          MODULES DE NIVEAU 3                          *)
(*                          -------------------                          *)
(*                                                                       *)
(*************************************************************************)
(*************************************************************************)
```

```
(*                                                              Ann. 141    *)
(*                          MODULE PARCOURS                                 *)
(*                                                                          *)
(**************************************************************************)

procedure parcours;
var position : integer;
    e        : integer;
    pe       : integer;
    finparc  : boolean;
begin
position := deb[res];
e := 0;
pe := -reseau[position];
finparc := false;
modnds (finparc,resultat,res,position,e,pe,resapp)
end;

(**************************************************************************)
(*                                                                        *)
(*                          MODULE REPRISE                                *)
(*                                                                        *).
(**************************************************************************)

procedure reprise;
var position : integer;
    e        : integer;
    pe       : integer;
    entree   : integer;
    arc      : integer;
    finparc  : boolean;
    finbr    : boolean;
begin
position := fin[res];
p := premier;
premier := premier^.suivant;
premier^.precedent := nil;
dispose (p);
entree := premier^.e;
position := reseau[position-entree];
resultat := false;
if (reseau[position] > 0) and (reseau[position] <= 1000)
then reprise (reseau[position],resultat,res);
if not resultat
then begin
     gestrep (res,position,resultat);
     if (resultat) and (reseau[position] >= 0)
     then begin
          arbre (position);
          finbr := false;
          finparc := false;
          modbr (finbr,finparc,resultat,position,e,pe,res);
          modnds (finparc,resultat,res,position,e,pe,resapp)
          end
     else
     if (resultat) and (reseau[position] < 0)
     then begin
          position := -reseau[position];
          e := rechent (position);
          position := position + e;
          pe := -reseau[position];
          finparc := false;
          modnds (finparc,resultat,res,position,e,pe,resapp)
          end
     end
else begin
```

```
      position := position + 1;
      position := -reseau[position];
      e := rechent(position);
      position := position + e;
      pe := -reseau[position];
      ndsproc(pe,e,resapp)
      end
end;

(*****************************************************************************)
(*                                                                           *)
(*                              SOUS-MODULES                                 *)
(*                                                                           *)
(* Ces modules ont ete separes des modules parcours-sous-reseau et reprise   *)
(* sous-reseau parceque les memes actions sont effectuees plusieurs fois a   *)
(* des endroits differents. En regroupant ainsi ces actions dans les modu-   *)
(* les modnds et modbr, on peut reduire la taille du code.                   *)
(*                                                                           *)
(*****************************************************************************)

procedure modnds;
var sortie : integer;
    finbr  : boolean;
begin
while not finparc do
      begin
      ndsproc (pe,e,resapp);
      if position = fin[res]
      then begin
           finparc := true;
           resultat := true
           end
      else begin
           sortie := premier^.s;
           position := reseau[position+sortie];
           finbr := false;
           modbr (finbr,finparc,resultat,position,e,pe,res)
           end
      end
end;

procedure modbr;
var arc : integer;
begin
while not finbr do
      begin
      arc := reseau[position];
      brlin (arc,resultat,res);
      if resultat
      then begin
           position := position + 1;
           arbre (position);
           if reseau[position] < 0
           then begin
                finbr := true;
                position := -reseau[position];
                e := rechent (position);
                position := position + e;
                pe := -reseau[position]
                end
           end
      else begin
           gestrep (res,position,resultat);
           if resultat
           then begin
                arbre (position);
```

```
                if reseau[position] < 0
                then begin
                     finbr := true;
                     position := -reseau[position];
                     e := rechent (position);
                     position := position + e;
                     pe := -reseau[position]
                     end
                end
           else begin
                finbr := true;
                finparc := true
                end
           end
      end
end;

(*************************************************************************)
(*                                                                       *)
(*                          MODULES DE NIVEAU 4                          *)
(*                          -------------------                          *)
(*                                                                       *)
(*************************************************************************)
(*************************************************************************)
(*                                                                       *)
(*                          MODULE AFFICHAGE-ARBRE                       *)
(*                                                                       *)
(*************************************************************************)

procedure arbre;
var arbinc : boolean;

procedure affich (res : integer ; position : integer ; var arbinc : boolean);
var pos    : integer;
    sortie : integer;
    finbr  : boolean;
    finres : boolean;
    arc    : elnt;
begin
finres := false;
p := p^.precedent;
pos := deb[res];
write ('( ');
while not (finres or arbinc) do
     begin
     if pos = fin[res]
     then begin
          finres := true
          end
     else begin
          while (p^.pe = 0) and (p <> premier) do
               begin
               p := p^.precedent
               end;
          sortie := p^.s;
          pos := reseau[pos + sortie];
          if (pos = position) and (p = premier)
          then arbinc := true
          else finbr := false;
          while not (finbr or arbinc) do
               begin
               if reseau[pos] = 0
               then write ('^ ')
               else
               if reseau[pos] <= 1000
               then affich (reseau[pos],position,arbinc)
```

```
                else
                if reseau[pos] <= 2000
                then begin
                     arc := ttl[reseau[pos] - 1000];
                     write (arc,' ')
                     end
                else
                if reseau[pos] <= 3000
                then begin
                     arc := ttv[reseau[pos] - 2000];
                     write (arc,' ')
                     end
                else begin
                     arc := ttg[reseau[pos] - 3000];
                     write (arc,' ')
                     end;
                pos := pos + 1;
                if (pos >= (position)) and (p = premier)
                then begin
                     arbinc := true
                     end;
                if reseau[pos] < 0
                then begin
                     finbr := true;
                     pos := -reseau[pos];
                     pos := pos + rechent (pos)
                     end
                end
           end;
      if not (finres or arbinc)
      then p := p^.precedent
      end;
if not arbinc
then begin
      arc := tnt[res];
      write (') * ',arc,' ')
      end
end;

begin
arbinc := false;
p := dernier;
affich (res,position,arbinc);
writeln
end;

(*****************************************************************************)
(*                                                                           *)
(*                          MODULE GESTION-REPRISES                          *)
(*                                                                           *)
(*****************************************************************************)

procedure gestrep;
var finrep : boolean;
    noeud  : boolean;
    ndent  : boolean;
    possib : boolean;
    sortie : integer;
    entree : integer;
begin
position := position - 1;
finrep := false;
while not finrep do
      begin
      noeud := false;
      while not noeud do
```

```
        begin
        if reseau[position] < 0
        then begin
             noeud := true
             end
        else
        if (0 < reseau[position]) and (reseau[position] <= 1000)
        then begin
             reprise (reseau[position],resultat,res);
             if resultat
             then begin
                  position := position + 1;
                  noeud := true;
                  finrep := true
                  end
             else begin
                  position := position - 1
                  end
             end
        else begin
             position := position - 1
             end
     end;
     if not finrep
     then begin
          position := -reseau[position];
          position := rechnd (position);
          if premier^.ps = -reseau[deb[res]]
          then ndent := true
          else ndent := false;
          if (premier^.suivant^.e = 0) and
             (premier^.suivant^.pe = 0)
          then possib := true
          else possib := false;
          if (ndent) and (not possib)
          then begin
               p := premier;
               premier := premier^.suivant;
               premier^.precedent := nil;
               dispose (p);
               finrep := true;
               resultat := false
               end
          else
          if possib
          then begin
               p := premier;
               premier := premier^.suivant;
               premier^.precedent := nil;
               dispose (p);
               sortie := premier^.s;
               position := reseau[position + sortie];
               finrep := true;
               resultat := true
               end
          else begin
               p := premier;
               premier := premier^.suivant;
               premier^.precedent := nil;
               dispose (p);
               entree := premier^.e;
               position := reseau[position - entree]
               end
          end
     end
end;
```

```
(*****************************************************************************)
(*                                                                           *)
(*                          MODULE BRANCHES-LINEAIRES                        *)
(*                                                                           *)
(*****************************************************************************)

procedure brlin;
type elt = array[1..bo2nt] of char;
var  nom : elt;

procedure branche (nom : elt ; var resultat : boolean);
var ch : char;
begin
writeln ('EST-CE QUE ',nom,' SUIT DANS LA PHRASE ?');
readln (ch);
if (ch = 'O') or (ch = 'o')
then resultat := true
else resultat := false
end;

begin
if arc < 0 then resultat := false
else
if arc = 0 then resultat := true
else
if arc <= 1000 then parcours (arc,resultat,res)
else
if arc <= 2000
then begin
     arc := arc - 1000;
     nom := ttl[arc];
     branche (nom,resultat)
     end
else
if arc <= 3000
then begin
     arc := arc - 2000;
     nom := ttv[arc];
     branche (nom,resultat)
     end
else begin
     arc := arc - 3000;
     nom := ttg[arc];
     branche (nom,resultat)
     end
end;

(*****************************************************************************)
(*                                                                           *)
(*                          MODULE NOEUDS-PROCEDURAUX                        *)
(*                                                                           *)
(*****************************************************************************)

procedure ndsproc;

procedure creel (proc : integer ; ns : integer);
begin
new (p);
p^.suivant := premier;
premier^.precedent := p;
premier := p;
premier^.ps := proc;
premier^.s := ns;
premier^.e := 0;
premier^.pe := 0;
```

```
premier^.precedent := nil;
end;

procedure noeud (proc : integer ; nbs : integer);
var i : integer;
begin
if nbs = 0
then creel (proc,nbs)
else for i := nbs downto 1 do creel (proc,i)
end;

begin
premier^.pe := pe;
premier^.e := e;
case pe of
     1 : noeud (1,3);
     2 : noeud (2,2);
     3 : noeud (3,2);
     4 : noeud (4,2);
     5 : noeud (5,4);
     6 : noeud (6,4);
     7 : noeud (7,1);
     8 : noeud (8,0);
     9 : noeud (9,3);
     10 : noeud (10,1);
     11 : noeud (11,3);
     12 : noeud (12,1);
     13 : noeud (13,3);
     14 : noeud (14,5);
     15 : noeud (15,3);
     16 : noeud (16,1);
     17 : noeud (17,0);
     18 : noeud (18,2);
     19 : noeud (19,0)
     end
end;

(*****************************************************************************)
(*                                                                           *)
(*                          PROGRAMME PRINCIPAL                              *)
(*                          -------------------                              *)
(*                                                                           *)
(*****************************************************************************)

begin
analyse
end.
```

```
(*****************************************************************************)
(*                                                                           *)
(*                            ANALYSEUR-SYNTAXIQUE                           *)
(*                            ====================                           *)
(*                                                                           *)
(*    Dans cette version de l'analyseur syntaxique, la definition des pro-   *)
(*    cedures associees aux differents noeuds proceduraux a ete separee de   *)
(*    la description des procedures de l'analyseur syntaxique proprement     *)
(*    dit. Elle se retrouvera dans un autre fichier, qui sera assemble avec  *)
(*    avec l'analyseur lors de l'edition des liens.                          *)
(*                                                                           *)
(*****************************************************************************)
(*                                                                           *)
(* AUTEUR                          : MOUSEL Pierre                           *)
(* DATE DERNIERE MODIFICATION : 13/12/82                                     *)
(*                                                                           *)
(*****************************************************************************)

program ansynt(input,output);
(*$h=20000*)

(*****************************************************************************)
(*                                                                           *)
(*                  DECLARATION DES DONNEES GLOBALES                         *)
(*                  --------------------------------                         *)
(*                                                                           *)
(*****************************************************************************)

const bo1nt = 20;
      bo2nt = 4;
      bo1tv = 20;
      bo2tv = 4;
      bo1tl = 30;
      bo2tl = 4;
      bo1tg = 20;
      bo2tg = 4;
      bores = 2000;
type  elnt     = array [1..bo2nt] of char;
      eltv     = array [1..bo2tv] of char;
      eltl     = array [1..bo2tl] of char;
      eltg     = array [1..bo2tg] of char;
      chaine   = string [bo2nt];
      pointeur = ^elt;
      elt      = record
                   suivant   : pointeur;
                   ps        : integer;
                   s         : integer;
                   e         : integer;
                   pe        : integer;
                   precedent : pointeur
                   end;
var   adc     : integer;
      nbnt    : integer;
      nbtv    : integer;
      nbtl    : integer;
      nbtg    : integer;
      res     : integer;
      sres    : chaine;
      tnt     : array [1..bo1nt] of elnt;
      ttv     : array [1..bo1tv] of eltv;
      ttl     : array [1..bo1tl] of eltl;
      ttg     : array [1..bo1tg] of eltg;
      deb     : array [1..bo1nt] of integer;
      fin     : array [1..bo1nt] of integer;
      reseau  : array [1..bores] of integer;
```

```
      p       : pointeur;
      premier : pointeur;
      dernier : pointeur;

(*************************************************************************)
(*                                                                       *)
(*                          LISTE DES RPOCEDURES                         *)
(*                          --------------------                         *)
(*                                                                       *)
(*************************************************************************)

procedure analyse ; forward;
procedure lecres  ; forward;
procedure anaphr  ; forward;
procedure parcours (    res      : integer;
                    var resultat : boolean;
                        resapp   : integer); forward;
procedure reprise  (    res      : integer;
                    var resultat : boolean;
                        resapp   : integer); forward;
procedure modnds   (    finparc  : boolean;
                    var resultat : boolean;
                        res      : integer;
                        position : integer;
                        e,pe     : integer;
                        resapp   : integer); forward;
procedure modbr    (var finbr    : boolean;
                    var finparc  : boolean;
                    var resultat : boolean;
                    var position : integer;
                    var e,pe     : integer;
                        res      : integer); forward;
procedure ndsproc  (pe : integer ; e : integer ; resapp : integer); forward;
procedure brlin    (    arc      : integer;
                    var resultat : boolean;
                        res      : integer); forward;
procedure gestrep  (    res      : integer ;
                    var position : integer ;
                    var resultat : boolean); forward;
procedure arbre    (position : integer); forward;

(*************************************************************************)
(*                                                                       *)
(*                          LES FONCTIONS UTILISEES                      *)
(*                          -----------------------                      *)
(*                                                                       *)
(*************************************************************************)

function  rechent (position : integer) : integer;
var i : integer;
begin
i := 0;
while (reseau[position + i]) > 0 do i := i + 1;
rechent := i
end;

function rechnd (position : integer) : integer;
begin
while reseau[position] > 0 do position := position - 1;
rechnd := position
end;

(*************************************************************************)
(*                                                                       *)
(*                          MODULES DE NIVEAU 1                          *)
(*                          -------------------                          *)
```

```
(*                                                                          *)
(****************************************************************************)
(****************************************************************************)
(*                                                                          *)
(*                          MODULE ANALYSEUR                                *)
(*                                                                          *)
(****************************************************************************)

procedure analyse;
begin
lecres;
anaphr
end;

(****************************************************************************)
(*                                                                          *)
(*                          MODULES DE NIVEAU 2                             *)
(*                          -------------------                             *)
(*                                                                          *)
(****************************************************************************)
(****************************************************************************)
(*                                                                          *)
(*                          MODULE LECTURE-RESEAU                           *)
(*                                                                          *)
(****************************************************************************)

procedure lecres;
var reponse : string[8];
    res     : text;
    i,j     : integer;
begin
writeln ('* VEUILLEZ ENTRER LE NOM DU FICHIER CONTENANT LE RNP.');
writeln ('* (AU MAXIMUM 8 CARACTERES.)');
readln (reponse);
reset (res,reponse);
readln (res,adc);
readln (res,nbnt);
for i := 1 to bo1nt do for j := 1 to bo2nt do read (res,tnt[i,j]);
readln (res);
readln (res,nbtv);
for i := 1 to bo1tv do for j := 1 to bo2tv do read (res,ttv[i,j]);
readln (res);
readln (res,nbtl);
for i := 1 to bo1tl do for j := 1 to bo2tl do read (res,ttl[i,j]);
readln (res);
readln (res,nbtg);
for i := 1 to bo1tg do for j := 1 to bo2tg do read (res,ttg[i,j]);
readln (res);
for i := 1 to bo1nt do readln (res,deb[i]);
for i := 1 to bo1nt do readln (res,fin[i]);
for i := 1 to bores do readln (res,reseau[i])
end;

(****************************************************************************)
(*                                                                          *)
(*                          MODULE ANALYSE-PHRASES                          *)
(*                                                                          *)
(****************************************************************************)

procedure anaphr;
var  reponse  : char;
     resultat : boolean;
     i        : integer;

function indice (sres : chaine) : integer;
var i,j    : integer;
```

```
    trouve : boolean;
begin
indice := 0;
for i := 1 to nbnt do
     begin
     trouve := true;
     for j := 1 to bo2nt do if sres[j] <> tnt [i,j] then trouve := false;
     if trouve then indice := i
     end
end;

begin
writeln ('* AVEZ VOUS UNE PHRASE A ANALYSER ?');
readln (reponse);
while (reponse = 'o') or (reponse = 'O') do
     begin
     writeln ('* VOUS AVEZ DECIDE D''ANALYSER UNE PHRASE.');
     writeln ('* QUEL EST LE PREMIER SOUS-RESEAU A PARCOURIR ?');
     for i := 1 to bo2nt do sres[i] := ' ';
     readln (sres);
     res := indice (sres);
     if res = 0
     then writeln ('* CE N''EST PAS UN NON-TERMINAL.')
     else begin
          new (p);
          premier := p;
          dernier := p;
          p^.suivant := nil;
          p^.precedent := nil;
          p^.ps := 0;
          p^.s := 0;
          p^.e := 0;
          p^.pe := 0;
          parcours (res,resultat,0);
          if resultat
          then begin
               writeln ('* LA PHRASE A ETE RECONNUE.');
               writeln ('* VOICI L''ARBRE SYNTXIQUE :');
               writeln;
               arbre(adc);
               writeln
               end
          else writeln ('* LA PHRASE N''A PAS ETE RECONNUE.')
          end;
     p := dernier;
     while p <> nil do
          begin
          dernier := dernier^.precedent;
          dispose (p);
          p := dernier
          end;
     writeln ('* AVEZ VOUS ENCORE UNE PHRASE A ANALYSER ?');
     readln (reponse)
     end
end;

(*************************************************************************)
(*                                                                       *)
(*                          MODULES DE NIVEAU 3                          *)
(*                          -------------------                          *)
(*                                                                       *)
(*************************************************************************)
(*************************************************************************)
(*                                                                       *)
(*                          MODULE PARCOURS                              *)
(*                                                                       *)
```

```
(*****************************************************************************)

procedure parcours;
var position : integer;
    e        : integer;
    pe       : integer;
    finparc  : boolean;
begin
position := deb[res];
e := 0;
pe := -reseau[position];
finparc := false;
modnds (finparc,resultat,res,position,e,pe,resapp)
end;

(*****************************************************************************)
(*                                                                           *)
(*                            MODULE REPRISE                                 *)
(*                                                                           *)
(*****************************************************************************)

procedure reprise;
var position : integer;
    e        : integer;
    pe       : integer;
    entree   : integer;
    arc      : integer;
    finparc  : boolean;
    finbr    : boolean;
begin
position := fin[res];
p := premier;
premier := premier^.suivant;
premier^.precedent := nil;
dispose (p);
entree := premier^.e;
position := reseau[position-entree];
resultat := false;
if (reseau[position] > 0) and (reseau[position] <= 1000)
then reprise (reseau[position],resultat,res);
if not resultat
then begin
     gestrep (res,position,resultat);
     if (resultat) and (reseau[position] >= 0)
     then begin
          arbre (position);
          finbr := false;
          finparc := false;
          modbr (finbr,finparc,resultat,position,e,pe,res);
          modnds (finparc,resultat,res,position,e,pe,resapp)
          end
     else
     if (resultat) and (reseau[position] < 0)
     then begin
          position := -reseau[position];
          e := rechent (position);
          position := position + e;
          pe := -reseau[position];
          finparc := false;
          modnds (finparc,resultat,res,position,e,pe,resapp)
          end
     end
else begin
     position := position + 1;
     position := -reseau[position];
     e := rechent(position);
```

```
     position := position + e;
     pe := -reseau[position];
     ndsproc(pe,e,resapp)
     end
end;

(*****************************************************************************)
(*                                                                           *)
(*                              SOUS-MODULES                                 *)
(*                                                                           *)
(* Ces modules ont ete separes des modules parcours-sous-reseau et reprise   *)
(* sous-reseau parceque les memes actions sont effectuees plusieurs fois a   *)
(* des endroits differents. En regroupant ainsi ces actions dans les modu-   *)
(* les modnds et modbr, on peut reduire la taille du code.                   *)
(*                                                                           *)
(*****************************************************************************)

procedure modnds;
var sortie : integer;
    finbr  : boolean;
begin
while not finparc do
     begin
     ndsproc (pe,e,resapp);
     if position = fin[res]
     then begin
          finparc := true;
          resultat := true
          end
     else begin
          sortie := premier^.s;
          position := reseau[position+sortie];
          finbr := false;
          modbr (finbr,finparc,resultat,position,e,pe,res)
          end
     end
end;

procedure modbr;
var arc : integer;
begin
while not finbr do
     begin
     arc := reseau[position];
     brlin (arc,resultat,res);
     if resultat
     then begin
          position := position + 1;
          arbre (position);
          if reseau[position] < 0
          then begin
               finbr := true;
               position := -reseau[position];
               e := rechent (position);
               position := position + e;
               pe := -reseau[position]
               end
          end
     else begin
          gestrep (res,position,resultat);
          if resultat
          then begin
               arbre (position);
               if reseau[position] < 0
               then begin
                    finbr := true;
```

```
                    position := -reseau[position];
                    e := rechent (position);
                    position := position + e;
                    pe := -reseau[position]
                    end
               end
          else begin
               finbr := true;
               finparc := true
               end
          end
     end
end;

(*************************************************************************)
(*                                                                       *)
(*                          MODULES DE NIVEAU 4                          *)
(*                          -------------------                          *)
(*                                                                       *)
(*************************************************************************)
(*************************************************************************)
(*                                                                       *)
(*                          MODULE AFFICHAGE-ARBRE                       *)
(*                                                                       *)
(*************************************************************************)

procedure arbre;
var arbinc : boolean;

procedure affich (res : integer ; position : integer ; var arbinc : boolean);
var pos    : integer;
    sortie : integer;
    finbr  : boolean;
    finres : boolean;
    arc    : elnt;
begin
finres := false;
p := p^.precedent;
pos := deb[res];
write ('( ');
while not (finres or arbinc) do
     begin
     if pos = fin[res]
     then begin
          finres := true
          end
     else begin
          while (p^.pe = 0) and (p <> premier) do
               begin
               p := p^.precedent
               end;
          sortie := p^.s;
          pos := reseau[pos + sortie];
          if (pos = position) and (p = premier)
          then arbinc := true
          else finbr := false;
          while not (finbr or arbinc) do
               begin
               if reseau[pos] = 0
               then write ('^ ')
               else
               if reseau[pos] <= 1000
               then affich (reseau[pos],position,arbinc)
               else
               if reseau[pos] <= 2000
               then begin
```

```
                    arc := ttl[reseau[pos] - 1000];
                    write (arc,' ')
                    end
               else
               if reseau[pos] <= 3000
               then begin
                    arc := ttv[reseau[pos] - 2000];
                    write (arc,' ')
                    end
               else begin
                    arc := ttg[reseau[pos] - 3000];
                    write (arc,' ')
                    end;
               pos := pos + 1;
               if (pos >= (position)) and (p = premier)
               then begin
                    arbinc := true
                    end;
               if reseau[pos] < 0
               then begin
                    finbr := true;
                    pos := -reseau[pos];
                    pos := pos + rechent (pos)
                    end
               end
          end;
     if not (finres or arbinc)
     then p := p^.precedent
     end;
if not arbinc
then begin
     arc := tnt[res];
     write (') * ',arc,' ')
     end
end;

begin
arbinc := false;
p := dernier;
affich (res,position,arbinc);
writeln
end;

(*****************************************************************************)
(*                                                                           *)
(*                          MODULE GESTION-REPRISES                          *)
(*                                                                           *)
(*****************************************************************************)

procedure gestrep;
var finrep : boolean;
    noeud  : boolean;
    ndent  : boolean;
    possib : boolean;
    sortie : integer;
    entree : integer;
begin
position := position - 1;
finrep := false;
while not finrep do
     begin
     noeud := false;
     while not noeud do
          begin
          if reseau[position] < 0
          then begin
```

```
                noeud := true
                end
          else
          if (0 < reseau[position]) and (reseau[position] <= 1000)
          then begin
                reprise (reseau[position],resultat,res);
                if resultat
                then begin
                     position := position + 1;
                     noeud := true;
                     finrep := true
                     end
                else begin
                     position := position - 1
                     end
                end
          else begin
                position := position - 1
                end
     end;
     if not finrep
     then begin
          position := -reseau[position];
          position := rechnd (position);
          if premier^.ps = -reseau[deb[res]]
          then ndent := true
          else ndent := false;
          if (premier^.suivant^.e = 0) and
             (premier^.suivant^.pe = 0)
          then possib := true
          else possib := false;
          if (ndent) and (not possib)
          then begin
                p := premier;
                premier := premier^.suivant;
                premier^.precedent := nil;
                dispose (p);
                finrep := true;
                resultat := false
                end
          else
          if possib
          then begin
                p := premier;
                premier := premier^.suivant;
                premier^.precedent := nil;
                dispose (p);
                sortie := premier^.s;
                position := reseau[position + sortie];
                finrep := true;
                resultat := true
                end
          else begin
                p := premier;
                premier := premier^.suivant;
                premier^.precedent := nil;
                dispose (p);
                entree := premier^.e;
                position := reseau[position - entree]
                end
          end
     end
end;

(*****************************************************************************)
(*                                                                           *)
```

```
(*                          MODULE BRANCHES-LINEAIRES                      *)
(*                                                                         *)
(*************************************************************************)

procedure brlin;
type elt = array[1..bo2nt] of char;
var  nom : elt;

procedure branche (nom : elt ; var resultat : boolean);
var ch : char;
begin
writeln ('EST-CE QUE ',nom,' SUIT DANS LA PHRASE ?');
readln (ch);
if (ch = 'O') or (ch = 'o')
then resultat := true
else resultat := false
end;

begin
if arc < 0 then resultat := false
else
if arc = 0 then resultat := true
else
if arc <= 1000 then parcours (arc,resultat,res)
else
if arc <= 2000
then begin
     arc := arc - 1000;
     nom := ttl[arc];
     branche (nom,resultat)
     end
else
if arc <= 3000
then begin
     arc := arc - 2000;
     nom := ttv[arc];
     branche (nom,resultat)
     end
else begin
     arc := arc - 3000;
     nom := ttg[arc];
     branche (nom,resultat)
     end
end;

(*************************************************************************)
(*                                                                       *)
(*                          MODULE NOEUDS-PROCEDURAUX                    *)
(*                                                                       *)
(* Ce module a ete defini comme externe pour rendre l'analyseur syntaxique *)
(* plus independant vis a vis du reseau particulier a parcourir.         *)
(*                                                                       *)
(*************************************************************************)

(*************************************************************************)
(*                                                                       *)
(*                          PROGRAMME PRINCIPAL                          *)
(*                          -------------------                          *)
(*                                                                       *)
(*************************************************************************)

begin
analyse
end.
```

```
(*****************************************************************************)
(*                                                                           *)
(*                            ANALYSEUR-SYNTAXIQUE                           *)
(*                            ====================                           *)
(*                                                                           *)
(*   Cette partie de l'analyseur syntaxique ne comprend que la description   *)
(*   des procedures associees aux differents noeuds proceduraux. Elle a ete  *)
(*   separee du code de l'analyseur lui meme pour rendre ce dernier plus     *)
(*   independant du reseau particulier a parcourir.                          *)
(*                                                                           *)
(*****************************************************************************)
(*                                                                           *)
(* AUTEUR                        : MOUSEL Pierre                             *)
(* DATE DERNIERE MODIFICATION    : 13/12/82                                  *)
(*                                                                           *)
(*****************************************************************************)

subprogram noeuds(input,output);
(*$h=20000*)

(*****************************************************************************)
(*                                                                           *)
(*                  DECLARATION DES DONNEES GLOBALES                         *)
(*                  --------------------------------                         *)
(*                                                                           *)
(*****************************************************************************)

const bo1nt = 20;
      bo2nt = 4;
      bo1tv = 20;
      bo2tv = 4;
      bo1tl = 30;
      bo2tl = 4;
      bo1tg = 20;
      bo2tg = 4;
      bores = 2000;
type  elnt     = array [1..bo2nt] of char;
      eltv     = array [1..bo2tv] of char;
      eltl     = array [1..bo2tl] of char;
      eltg     = array [1..bo2tg] of char;
      chaine   = string [bo2nt];
      pointeur = ^elt;
      elt      = record
                   suivant   : pointeur;
                   ps        : integer;
                   s         : integer;
                   e         : integer;
                   pe        : integer;
                   precedent : pointeur
                   end;
var   adc     : integer;
      nbnt    : integer;
      nbtv    : integer;
      nbtl    : integer;
      nbtg    : integer;
      res     : integer;
      sres    : chaine;
      tnt     : array [1..bo1nt] of elnt;
      ttv     : array [1..bo1tv] of eltv;
      ttl     : array [1..bo1tl] of eltl;
      ttg     : array [1..bo1tg] of eltg;
      deb     : array [1..bo1nt] of integer;
      fin     : array [1..bo1nt] of integer;
      reseau  : array [1..bores] of integer;
      p       : pointeur;
```

```
      premier : pointeur;
      dernier : pointeur;

(*************************************************************************)
(*                                                                       *)
(*                          LISTE DES RPOCEDURES                         *)
(*                          --------------------                         *)
(*                                                                       *)
(*************************************************************************)

procedure analyse ; forward;
procedure lecres  ; forward;
procedure anaphr  ; forward;
procedure parcours (    res       : integer;
                    var resultat  : boolean;
                        resapp    : integer); forward;
procedure reprise  (    res       : integer;
                    var resultat  : boolean;
                        resapp    : integer); forward;
procedure modnds   (    finparc   : boolean;
                    var resultat  : boolean;
                        res       : integer;
                        position  : integer;
                        e,pe      : integer;
                        resapp    : integer); forward;
procedure modbr    (var finbr     : boolean;
                    var finparc   : boolean;
                    var resultat  : boolean;
                    var position  : integer;
                    var e,pe      : integer;
                        res       : integer); forward;
procedure ndsproc  (pe : integer ; e : integer ; resapp : integer); forward;
procedure brlin    (    arc       : integer;
                    var resultat  : boolean;
                        res       : integer); forward;
procedure gestrep  (    res       : integer ;
                    var position  : integer ;
                    var resultat  : boolean); forward;
procedure arbre    (position : integer); forward;

(*************************************************************************)
(*                                                                       *)
(*                          MODULE NOEUDS-PROCEDURAUX                    *)
(*                                                                       *)
(*************************************************************************)

procedure ndsproc;

procedure creel (proc : integer ; ns : integer);
begin
new (p);
p^.suivant := premier;
premier^.precedent := p;
premier := p;
premier^.ps := proc;
premier^.s := ns;
premier^.e := 0;
premier^.pe := 0;
premier^.precedent := nil;
end;

procedure noeud (proc : integer ; nbs : integer);
var i : integer;
begin
if nbs = 0
then creel (proc,nbs)
```

```
else for i := nbs downto 1 do creel (proc,i)
end;

begin
premier^.pe := pe;
premier^.e := e;
case pe of
     1 : noeud (1,3);
     2 : noeud (2,2);
     3 : noeud (3,2);
     4 : noeud (4,2);
     5 : noeud (5,4);
     6 : noeud (6,4);
     7 : noeud (7,1);
     8 : noeud (8,0);
     9 : noeud (9,3);
     10 : noeud (10,1);
     11 : noeud (11,3);
     12 : noeud (12,1);
     13 : noeud (13,3);
     14 : noeud (14,5);
     15 : noeud (15,3);
     16 : noeud (16,1);
     17 : noeud (17,0);
     18 : noeud (18,2);
     19 : noeud (19,0)
     end
end.
```

```
(*************************************************************************)
(*                                                                       *)
(*                          ANALYSEUR-SYNTAXIQUE                         *)
(*                          ====================                         *)
(*                                                                       *)
(*   Cette version de l'analyseur syntaxique comprend la definition des  *)
(*   noeuds-proceduraux dans la definition des procedures de l'analyseur *)
(*   meme. Si donc on change de reseau, il faut modifier la procedure    *)
(*   'ndsproc' de l'analyseur. Pour eviter cet ennui, il existe une deu- *)
(*   xieme version de cet analyseur, ou le module noeuds-proceduraux est *)
(*   declare externe. Cette version peut etre trouvee sur le fichier :   *)
(*                                                                       *)
(*                             ansynt4                                   *)
(*                                                                       *)
(*   De plus cette version a introduit un indeterminisme supplementaire  *)
(*   par rapport aux versions sur les fichiers ansynt1 et ansynt2, ce qui *)
(*   signifie que dans cette version, plusieurs mots peuvent etre reconnus *)
(*   par classe grammaticale, avec des scores differents ou egaux.       *)
(*                                                                       *)
(*************************************************************************)
(*                                                                       *)
(* AUTEUR                       : MOUSEL Pierre                          *)
(* DATE DERNIERE MODIFICATION   : 13/12/82                               *)
(*                                                                       *)
(*************************************************************************)

program ansynt(input,output);
(*$h=40000*)

(*************************************************************************)
(*                                                                       *)
(*                DECLARATION DES DONNEES GLOBALES                       *)
(*                --------------------------------                       *)
(*                                                                       *)
(*************************************************************************)

const bo1nt = 20;
      bo2nt = 4;
      bo1tv = 20;
      bo2tv = 4;
      bo1tl = 30;
      bo2tl = 4;
      bo1tg = 20;
      bo2tg = 4;
      bores = 2000;
      longm = 20;
type  elnt     = array [1..bo2nt] of char;
      eltv     = array [1..bo2tv] of char;
      eltl     = array [1..bo2tl] of char;
      eltg     = array [1..bo2tg] of char;
      chaine   = string [bo2nt];
      pointelt = ^elt;
      pointmot = ^mots;
      elt      = record
                   suivant   : pointelt;
                   ps        : integer;
                   s         : integer;
                   e         : integer;
                   pe        : integer;
                   premot    : pointmot;
                   precedent : pointelt
                   end;
      mots     = record
                   suivant   : pointmot;
                   mot       : string[longm];
```

```
                  score     : integer;
                  position  : integer;
                  pelt      : pointelt;
                  precedent : pointmot
                  end;
var   adc      : integer;
      nbnt     : integer;
      nbtv     : integer;
      nbtl     : integer;
      nbtg     : integer;
      res      : integer;
      sres     : chaine;
      tnt      : array [1..bolnt] of elnt;
      ttv      : array [1..boltv] of eltv;
      ttl      : array [1..boltl] of eltl;
      ttg      : array [1..boltg] of eltg;
      deb      : array [1..bolnt] of integer;
      fin      : array [1..bolnt] of integer;
      reseau   : array [1..bores] of integer;
      p        : pointelt;
      premier  : pointelt;
      dernier  : pointelt;
      pm       : pointmot;
      pmint    : pointmot;
      dermot   : pointmot;
      premot   : pointmot;

(*****************************************************************************)
(*                                                                           *)
(*                          LISTE DES RPOCEDURES                             *)
(*                          --------------------                             *)
(*                                                                           *)
(*****************************************************************************)

procedure analyse ; forward;
procedure lecres  ; forward;
procedure anaphr  ; forward;
procedure parcours (    res      : integer;
                    var resultat : boolean;
                        resapp   : integer); forward;
procedure reprise  (    res      : integer;
                    var resultat : boolean;
                        resapp   : integer); forward;
procedure modnds   (    finparc  : boolean;
                    var resultat : boolean;
                        res      : integer;
                        position : integer;
                        e,pe     : integer;
                        resapp   : integer); forward;
procedure modbr    (var finbr    : boolean;
                    var finparc  : boolean;
                    var resultat : boolean;
                    var position : integer;
                    var e,pe     : integer;
                        res      : integer); forward;
procedure ndsproc  (pe : integer ; e : integer ; resapp : integer); forward;
procedure brlin    (    arc      : integer;
                    var resultat : boolean;
                        res      : integer;
                        position : integer); forward;
procedure repbr    (    position : integer;
                    var resultat : boolean); forward;
procedure gestrep  (    res      : integer ;
                    var position : integer ;
                    var resultat : boolean); forward;
procedure arbre    (position : integer); forward;
```

```
(*************************************************************************)
(*                                                                       *)
(*                          LES FONCTIONS UTILISEES                      *)
(*                          -----------------------                      *)
(*                                                                       *)
(*************************************************************************)

function  rechent (position : integer) : integer;
var i : integer;
begin
i := 0;
while (reseau[position + i]) > 0 do i := i + 1;
rechent := i
end;

function rechnd (position : integer) : integer;
begin
while reseau[position] > 0 do position := position - 1;
rechnd := position
end;

(*************************************************************************)
(*                                                                       *)
(*                          MODULES DE NIVEAU 1                          *)
(*                          -------------------                          *)
(*                                                                       *)
(*************************************************************************)
(*************************************************************************)
(*                                                                       *)
(*                          MODULE ANALYSEUR                             *)
(*                                                                       *)
(*************************************************************************)

procedure analyse;
begin
lecres;
anaphr
end;

(*************************************************************************)
(*                                                                       *)
(*                          MODULES DE NIVEAU 2                          *)
(*                          -------------------                          *)
(*                                                                       *)
(*************************************************************************)
(*************************************************************************)
(*                                                                       *)
(*                          MODULE LECTURE-RESEAU                        *)
(*                                                                       *)
(*************************************************************************)

procedure lecres;
var reponse : string[8];
    res     : text;
    i,j     : integer;
begin
writeln ('* VEUILLEZ ENTRER LE NOM DU FICHIER CONTENANT LE RNP.');
writeln ('* (AU MAXIMUM 8 CARACTERES.)');
readln (reponse);
reset (res,reponse);
readln (res,adc);
readln (res,nbnt);
for i := 1 to bo1nt do for j := 1 to bo2nt do read (res,tnt[i,j]);
readln (res);
readln (res,nbtv);
```

```
for i := 1 to bo1tv do for j := 1 to bo2tv do read (res,ttv[i,j]);
readln (res);
readln (res,nbtl);
for i := 1 to bo1tl do for j := 1 to bo2tl do read (res,ttl[i,j]);
readln (res);
readln (res,nbtg);
for i := 1 to bo1tg do for j := 1 to bo2tg do read (res,ttg[i,j]);
readln (res);
for i := 1 to bo1nt do readln (res,deb[i]);
for i := 1 to bo1nt do readln (res,fin[i]);
for i := 1 to bores do readln (res,reseau[i])
end;

(*****************************************************************************)
(*                                                                           *)
(*                          MODULE ANALYSE-PHRASES                           *)
(*                                                                           *)
(*****************************************************************************)

procedure anaphr;
var  reponse  : char;
     resultat : boolean;
     i        : integer;

function indice (sres : chaine) : integer;
var i,j    : integer;
    trouve : boolean;
begin
indice := 0;
for i := 1 to nbnt do
     begin
     trouve := true;
     for j := 1 to bo2nt do if sres[j] <> tnt [i,j] then trouve := false;
     if trouve then indice := i
     end
end;

begin
writeln ('* AVEZ VOUS UNE PHRASE A ANALYSER ?');
readln (reponse);
while (reponse = 'o') or (reponse = 'O') do
     begin
     writeln ('* VOUS AVEZ DECIDE D''ANALYSER UNE PHRASE.');
     writeln ('* QUEL EST LE PREMIER SOUS-RESEAU A PARCOURIR ?');
     for i := 1 to bo2nt do sres[i] := ' ';
     readln (sres);
     res := indice (sres);
     if res = 0
     then writeln ('* CE N''EST PAS UN NON-TERMINAL.')
     else begin
          new (pm);
          premot := pm;
          dermot := pm;
          pm^.suivant := nil;
          pm^.precedent := nil;
          for i := 1 to longm do pm^.mot[i] := ' ';
          pm^.score := 0;
          pm^.position := 0;
          new (p);
          premier := p;
          dernier := p;
          premot^.pelt := p;
          p^.suivant := nil;
          p^.precedent := nil;
          p^.premot := premot;
          p^.ps := 0;
```

```
          p^.s := 0;
          p^.e := 0;
          p^.pe := 0;
          parcours (res,resultat,0);
          if resultat
          then begin
               writeln ('* LA PHRASE A ETE RECONNUE.');
               writeln ('* VOICI L''ARBRE SYNTXIQUE :');
               writeln;
               arbre(adc);
               writeln
               end
          else writeln ('* LA PHRASE N''A PAS ETE RECONNUE.')
          end;
     writeln ('* AVEZ VOUS ENCORE UNE PHRASE A ANALYSER ?');
     pm := dermot;
     p := dernier;
     while pm <> nil do
          begin
          dermot := dermot^.precedent;
          dispose (pm);
          pm := dermot
          end;
     while p <> nil do
          begin
          dernier := dernier^.precedent;
          dispose (p);
          p := dernier
          end;
     readln (reponse)
     end
end;

(*****************************************************************************)
(*                                                                           *)
(*                          MODULES DE NIVEAU 3                              *)
(*                          -------------------                              *)
(*                                                                           *)
(*****************************************************************************)
(*****************************************************************************)
(*                                                                           *)
(*                          MODULE PARCOURS                                  *)
(*                                                                           *)
(*****************************************************************************)

procedure parcours;
var position : integer;
    e        : integer;
    pe       : integer;
    finparc  : boolean;
begin
position := deb[res];
e := 0;
pe := -reseau[position];
finparc := false;
modnds (finparc,resultat,res,position,e,pe,resapp)
end;

(*****************************************************************************)
(*                                                                           *)
(*                          MODULE REPRISE                                   *)
(*                                                                           *)
(*****************************************************************************)

procedure reprise;
var position : integer;
```

```
    e         : integer;
    pe        : integer;
    entree    : integer;
    arc       : integer;
    finparc   : boolean;
    finbr     : boolean;
begin
position := fin[res];
p := premier;
premier := premier^.suivant;
premier^.precedent := nil;
dispose (p);
entree := premier^.e;
position := reseau[position-entree];
resultat := false;
if (reseau[position] > 0) and (reseau[position] <= 1000)
then reprise (reseau[position],resultat,res)
else repbr (position,resultat);
if not resultat
then begin
     gestrep (res,position,resultat);
     if (resultat) and (reseau[position] >= 0)
     then begin
          arbre (position);
          finbr := false;
          finparc := false;
          modbr (finbr,finparc,resultat,position,e,pe,res);
          modnds (finparc,resultat,res,position,e,pe,resapp)
          end
     else
     if (resultat) and (reseau[position] < 0)
     then begin
          position := -reseau[position];
          e := rechent (position);
          position := position + e;
          pe := -reseau[position];
          finparc := false;
          modnds (finparc,resultat,res,position,e,pe,resapp)
          end
     end
else begin
     position := position + 1;
     position := -reseau[position];
     e := rechent(position);
     position := position + e;
     pe := -reseau[position];
     ndsproc(pe,e,resapp)
     end
end;

(*****************************************************************************)
(*                                                                           *)
(*                              SOUS-MODULES                                 *)
(*                                                                           *)
(* Ces modules ont ete separes des modules parcours-sous-reseau et reprise   *)
(* sous-reseau parceque les memes actions sont effectuees plusieurs fois a   *)
(* des endroits differents. En regroupant ainsi ces actions dans les modu-   *)
(* les modnds et modbr, on peut reduire la taille du code.                   *)
(*                                                                           *)
(*****************************************************************************)

procedure modnds;
var sortie : integer;
    finbr  : boolean;
begin
while not finparc do
```

```
     begin
     ndsproc (pe,e,resapp);
     if position = fin[res]
     then begin
          finparc := true;
          resultat := true
          end
     else begin
          sortie := premier^.s;
          position := reseau[position+sortie];
          finbr := false;
          modbr (finbr,finparc,resultat,position,e,pe,res)
          end
     end
end;

procedure modbr;
var arc : integer;
begin
while not finbr do
     begin
     arc := reseau[position];
     brlin (arc,resultat,res,position);
     if resultat
     then begin
          position := position + 1;
          arbre (position);
          if reseau[position] < 0
          then begin
               finbr := true;
               position := -reseau[position];
               e := rechent (position);
               position := position + e;
               pe := -reseau[position]
               end
          end
     else begin
          gestrep (res,position,resultat);
          if resultat
          then begin
               arbre (position);
               if reseau[position] < 0
               then begin
                    finbr := true;
                    position := -reseau[position];
                    e := rechent (position);
                    position := position + e;
                    pe := -reseau[position]
                    end
               end
          else begin
               finbr := true;
               finparc := true
               end
          end
     end
end;

(*************************************************************************)
(*                                                                       *)
(*                          MODULES DE NIVEAU 4                          *)
(*                          -------------------                          *)
(*                                                                       *)
(*************************************************************************)
(*************************************************************************)
(*                                                                       *)
```

```
(*                                MODULE AFFICHAGE-ARBRE                    Ann. 168    *)
(*                                                                                      *)
(****************************************************************************************)

procedure arbre;
var arbinc : boolean;

procedure affich (res : integer ; position : integer ; var arbinc : boolean);
var pos    : integer;
    sortie : integer;
    finbr  : boolean;
    finres : boolean;
    arc    : elnt;
begin
finres := false;
p := p^.precedent;
pos := deb[res];
write ('( ');
while not (finres or arbinc) do
     begin
     if pos = fin[res]
     then begin
          finres := true
          end
     else begin
          while (p^.pe = 0) and (p <> premier) do
               begin
               p := p^.precedent
               end;
          sortie := p^.s;
          pos := reseau[pos + sortie];
          if (pos = position) and (p = premier)
          then arbinc := true
          else finbr := false;
          while not (finbr or arbinc) do
               begin
               if reseau[pos] = 0
               then write ('^ ')
               else
               if reseau[pos] <= 1000
               then affich (reseau[pos],position,arbinc)
               else
               if reseau[pos] <= 2000
               then begin
                    arc := ttl[reseau[pos] - 1000];
                    write (arc,' ')
                    end
               else
               if reseau[pos] <= 3000
               then begin
                    arc := ttv[reseau[pos] - 2000];
                    write (arc,' ')
                    end
               else begin
                    arc := ttg[reseau[pos] - 3000];
                    write (arc,' ')
                    end;
               pos := pos + 1;
               if (pos >= (position)) and (p = premier)
               then begin
                    arbinc := true
                    end;
               if reseau[pos] < 0
               then begin
                    finbr := true;
                    pos := -reseau[pos];
```

```
                    pos := pos + rechent (pos)
                    end
               end
          end;
     if not (finres or arbinc)
     then p := p^.precedent
     end;
if not arbinc
then begin
     arc := tnt[res];
     write (') * ',arc,' ')
     end
end;

begin
arbinc := false;
p := dernier;
affich (res,position,arbinc);
writeln
end;

(*****************************************************************************)
(*                                                                           *)
(*                          MODULE GESTION-REPRISES                          *)
(*                                                                           *)
(*****************************************************************************)

procedure gestrep;
var finrep : boolean;
    noeud  : boolean;
    ndent  : boolean;
    possib : boolean;
    sortie : integer;
    entree : integer;
begin
position := position - 1;
finrep := false;
while not finrep do
     begin
     noeud := false;
     while not noeud do
          begin
          if reseau[position] < 0
          then begin
               noeud := true
               end
          else
          if (0 < reseau[position]) and (reseau[position] <= 1000)
          then begin
               reprise (reseau[position],resultat,res);
               if resultat
               then begin
                    position := position + 1;
                    noeud := true;
                    finrep := true
                    end
               else begin
                    position := position - 1
                    end
               end
          else begin
               repbr (position,resultat);
               if resultat
               then begin
                    position := position + 1;
                    noeud := true;
```

```
                    finrep := true
                    end
               else begin
                    position := position - 1
                    end
               end
     end;
     if not finrep
     then begin
          position := -reseau[position];
          position := rechnd (position);
          if premier^.ps = -reseau[deb[res]]
          then ndent := true
          else ndent := false;
          if (premier^.suivant^.e = 0) and
             (premier^.suivant^.pe = 0)
          then possib := true
          else possib := false;
          if (ndent) and (not possib)
          then begin
               p := premier;
               premier := premier^.suivant;
               premier^.precedent := nil;
               dispose (p);
               finrep := true;
               resultat := false
               end
          else
          if possib
          then begin
               p := premier;
               premier := premier^.suivant;
               premier^.precedent := nil;
               dispose (p);
               sortie := premier^.s;
               position := reseau[position + sortie];
               finrep := true;
               resultat := true
               end
          else begin
               p := premier;
               premier := premier^.suivant;
               premier^.precedent := nil;
               dispose (p);
               entree := premier^.e;
               position := reseau[position - entree]
               end
          end
     end
end;

(*****************************************************************************)
(*                                                                           *)
(*                          MODULE BRANCHES-LINEAIRES                        *)
(*                                                                           *)
(*****************************************************************************)

procedure brlin;
type elt = array[1..bo2nt] of char;
var  nom : elt;

procedure branche (nom : elt ; var resultat : boolean ; position : integer);
var ch      : char;
    motvide : boolean;
    i       : integer;
    score   : integer;
```

```
    mot     : string[longm];
begin
writeln ('EST-CE QUE ',nom,' SUIT DANS LA PHRASE ?');
readln (ch);
resultat := false;
if (ch = 'O') or (ch = 'o')
then begin
     for i := 1 to longm do mot[i] := ' ';
     writeln ('VOUS DEVEZ INTRODUIRE LES ',nom,' RECONNUS ET LES SCORES.');
     writeln ('MOT :');
     readln (mot);
     motvide := true;
     for i := 1 to longm do if mot[i] <> ' ' then motvide := false;
     while not motvide do
          begin
          resultat := true;
          writeln ('SCORE :');
          readln (score);
          if (premier <> premot^.pelt) or
             (position <> premot^.position) or
             ((premier = premot^.pelt) and
              (position = premot^.position) and
              (score >= premot^.score))
          then begin
               new (pm);
               premier^.premot := pm;
               pm^.suivant := premot;
               premot^.precedent := pm;
               premot := pm;
               premot^.precedent := nil;
               premot^.mot := mot;
               premot^.score := score;
               premot^.position := position;
               premot^.pelt := premier
               end
          else begin
               pmint := premot;
               while (position = pmint^.position) and
                     (premier = pmint^.pelt) and
                     (score < pmint^.score) do
                    begin
                    pmint := pmint^.suivant
                    end;
               new (pm);
               pm^.suivant := pmint;
               pm^.precedent := pmint^.precedent;
               pmint^.precedent^.suivant := pm;
               pmint^.precedent := pm;
               pm^.mot := mot;
               pm^.score := score;
               pm^.position := position;
               pm^.pelt := premier
               end;
          writeln ('MOT :');
          for i := 1 to longm do mot[i] := ' ';
          readln (mot);
          motvide := true;
          for i := 1 to longm do if mot[i] <> ' ' then motvide := false
          end
     end
end;

begin
if arc < 0 then resultat := false
else
if arc = 0 then resultat := true
```

```
else
if arc <= 1000 then parcours (arc,resultat,res)
else
if arc <= 2000
then begin
     arc := arc - 1000;
     nom := ttl[arc];
     branche (nom,resultat,position)
     end
else
if arc <= 3000
then begin
     arc := arc - 2000;
     nom := ttv[arc];
     branche (nom,resultat,position)
     end
else begin
     arc := arc - 3000;
     nom := ttg[arc];
     branche (nom,resultat,position)
     end
end;

(*****************************************************************************)
(*                                                                           *)
(*                          MODULE REPRISE-BRANCHES                          *)
(*                                                                           *)
(*****************************************************************************)

procedure repbr;
begin
if (premier = premot^.suivant^.pelt) and
   (position = premot^.suivant^.position)
then begin
     resultat := true;
     pm := premot;
     premot := premot^.suivant;
     premot^.precedent := nil;
     premier^.premot := premot;
     dispose (pm)
     end
else begin
     resultat := false;
     if (premier = premot^.suivant^.pelt)
     then begin
          pm := premot;
          premot := premot^.suivant;
          premot^.precedent := nil;
          premier^.premot := premot;
          dispose (pm)
          end
     else begin
          pm := premot;
          premot := premot^.suivant;
          premot^.precedent := nil;
          premier^.premot := nil;
          dispose (pm)
          end
     end
end;

(*****************************************************************************)
(*                                                                           *)
(*                        MODULE NOEUDS-PROCEDURAUX                          *)
(*                                                                           *)
(*****************************************************************************)
```

```
procedure ndsproc;

procedure creel (proc : integer ; ns : integer);
begin
new (p);
p^.suivant := premier;
premier^.precedent := p;
premier := p;
premier^.ps := proc;
premier^.s := ns;
premier^.e := 0;
premier^.pe := 0;
premier^.premot := nil;
premier^.precedent := nil;
end;

procedure noeud (proc : integer ; nbs : integer);
var i : integer;
begin
if nbs = 0
then creel (proc,nbs)
else for i := nbs downto 1 do creel (proc,i)
end;

begin
premier^.pe := pe;
premier^.e := e;
case pe of
     1 : noeud (1,3);
     2 : noeud (2,2);
     3 : noeud (3,2);
     4 : noeud (4,2);
     5 : noeud (5,4);
     6 : noeud (6,4);
     7 : noeud (7,1);
     8 : noeud (8,0);
     9 : noeud (9,3);
     10 : noeud (10,1);
     11 : noeud (11,3);
     12 : noeud (12,1);
     13 : noeud (13,3);
     14 : noeud (14,5);
     15 : noeud (15,3);
     16 : noeud (16,1);
     17 : noeud (17,0);
     18 : noeud (18,2);
     19 : noeud (19,0)
     end
end;

(*************************************************************************)
(*                                                                       *)
(*                        PROGRAMME PRINCIPAL                            *)
(*                        -------------------                            *)
(*                                                                       *)
(*************************************************************************)

begin
analyse
end.
```

```
(*****************************************************************************)
(*                                                                           *)
(*                            ANALYSEUR-SYNTAXIQUE                           *)
(*                            ====================                           *)
(*                                                                           *)
(*   Dans cette version de l'analyseur syntaxique, la description des pro-   *)
(*   cedures associees aux differents noeuds proceduraux a ete separee du    *)
(*   code de l'analyseur syntaxique lui meme. On la retrouvera sur un fi-    *)
(*   chier qui sera assemble avec l'analyseur syntaxique au moment de l'e-   *)
(*   dition des liens.                                                       *)
(*   De plus cette version a introduit un indeterminisme supplementaire      *)
(*   par rapport aux versions sur les fichiers ansynt1 et ansynt2, ce qui    *)
(*   signifie que dans cette version, plusieurs mots peuvent etre reconnus   *)
(*   par classe grammaticale, avec des scores differents ou egaux.           *)
(*                                                                           *)
(*****************************************************************************)
(*                                                                           *)
(* AUTEUR                          : MOUSEL Pierre                           *)
(* DATE DERNIERE MODIFICATION      : 13/12/82                                *)
(*                                                                           *)
(*****************************************************************************)

program ansynt(input,output);
(*$h=40000*)

(*****************************************************************************)
(*                                                                           *)
(*                  DECLARATION DES DONNEES GLOBALES                         *)
(*                  --------------------------------                         *)
(*                                                                           *)
(*****************************************************************************)

const bo1nt = 20;
      bo2nt = 4;
      bo1tv = 20;
      bo2tv = 4;
      bo1tl = 30;
      bo2tl = 4;
      bo1tg = 20;
      bo2tg = 4;
      bores = 2000;
      longm = 20;
type  elnt     = array [1..bo2nt] of char;
      eltv     = array [1..bo2tv] of char;
      eltl     = array [1..bo2tl] of char;
      eltg     = array [1..bo2tg] of char;
      chaine   = string [bo2nt];
      pointelt = ^elt;
      pointmot = ^mots;
      elt      = record
                   suivant   : pointelt;
                   ps        : integer;
                   s         : integer;
                   e         : integer;
                   pe        : integer;
                   premot    : pointmot;
                   precedent : pointelt
                   end;
      mots     = record
                   suivant   : pointmot;
                   mot       : string[longm];
                   score     : integer;
                   position  : integer;
                   pelt      : pointelt;
                   precedent : pointmot
```

```
                end;
var   adc     : integer;
      nbnt    : integer;
      nbtv    : integer;
      nbtl    : integer;
      nbtg    : integer;
      res     : integer;
      sres    : chaine;
      tnt     : array [1..bolnt] of elnt;
      ttv     : array [1..boltv] of eltv;
      ttl     : array [1..boltl] of eltl;
      ttg     : array [1..boltg] of eltg;
      deb     : array [1..bolnt] of integer;
      fin     : array [1..bolnt] of integer;
      reseau  : array [1..bores] of integer;
      p       : pointelt;
      premier : pointelt;
      dernier : pointelt;
      pm      : pointmot;
      pmint   : pointmot;
      dermot  : pointmot;
      premot  : pointmot;

(*************************************************************************)
(*                                                                       *)
(*                          LISTE DES RPOCEDURES                         *)
(*                          --------------------                         *)
(*                                                                       *)
(*************************************************************************)

procedure analyse ; forward;
procedure lecres  ; forward;
procedure anaphr  ; forward;
procedure parcours (    res      : integer;
                    var resultat : boolean;
                        resapp   : integer); forward;
procedure reprise  (    res      : integer;
                    var resultat : boolean;
                        resapp   : integer); forward;
procedure modnds   (    finparc  : boolean;
                    var resultat : boolean;
                        res      : integer;
                        position : integer;
                        e,pe     : integer;
                        resapp   : integer); forward;
procedure modbr    (var finbr    : boolean;
                    var finparc  : boolean;
                    var resultat : boolean;
                    var position : integer;
                    var e,pe     : integer;
                        res      : integer); forward;
procedure ndsproc  (pe : integer ; e : integer ; resapp : integer); forward;
procedure brlin    (    arc      : integer;
                    var resultat : boolean;
                        res      : integer;
                        position : integer); forward;
procedure repbr    (    position : integer;
                    var resultat : boolean); forward;
procedure gestrep  (    res      : integer ;
                    var position : integer ;
                    var resultat : boolean); forward;
procedure arbre    (position : integer); forward;

(*************************************************************************)
(*                                                                       *)
(*                          LES FONCTIONS UTILISEES                      *)
```

```
(*                          ----------------------                          *)
(*                                                                          *)
(****************************************************************************)

function  rechent (position : integer) : integer;
var i : integer;
begin
i := 0;
while (reseau[position + i]) > 0 do i := i + 1;
rechent := i
end;

function rechnd (position : integer) : integer;
begin
while reseau[position] > 0 do position := position - 1;
rechnd := position
end;

(****************************************************************************)
(*                                                                          *)
(*                          MODULES DE NIVEAU 1                             *)
(*                          -------------------                             *)
(*                                                                          *)
(****************************************************************************)
(****************************************************************************)
(*                                                                          *)
(*                          MODULE ANALYSEUR                                *)
(*                                                                          *)
(****************************************************************************)

procedure analyse;
begin
lecres;
anaphr
end;

(****************************************************************************)
(*                                                                          *)
(*                          MODULES DE NIVEAU 2                             *)
(*                          -------------------                             *)
(*                                                                          *)
(****************************************************************************)
(****************************************************************************)
(*                                                                          *)
(*                          MODULE LECTURE-RESEAU                           *)
(*                                                                          *)
(****************************************************************************)

procedure lecres;
var reponse : string[8];
    res     : text;
    i,j     : integer;
begin
writeln ('* VEUILLEZ ENTRER LE NOM DU FICHIER CONTENANT LE RNP.');
writeln ('* (AU MAXIMUM 8 CARACTERES.)');
readln (reponse);
reset (res,reponse);
readln (res,adc);
readln (res,nbnt);
for i := 1 to bo1nt do for j := 1 to bo2nt do read (res,tnt[i,j]);
readln (res);
readln (res,nbtv);
for i := 1 to bo1tv do for j := 1 to bo2tv do read (res,ttv[i,j]);
readln (res);
readln (res,nbtl);
for i := 1 to bo1tl do for j := 1 to bo2tl do read (res,ttl[i,j]);
```

```
readln (res);
readln (res,nbtg);
for i := 1 to bo1tg do for j := 1 to bo2tg do read (res,ttg[i,j]);
readln (res);
for i := 1 to bo1nt do readln (res,deb[i]);
for i := 1 to bo1nt do readln (res,fin[i]);
for i := 1 to bores do readln (res,reseau[i])
end;

(*****************************************************************************)
(*                                                                           *)
(*                          MODULE ANALYSE-PHRASES                           *)
(*                                                                           *)
(*****************************************************************************)

procedure anaphr;
var  reponse  : char;
     resultat : boolean;
     i        : integer;

function indice (sres : chaine) : integer;
var i,j    : integer;
    trouve : boolean;
begin
indice := 0;
for i := 1 to nbnt do
     begin
     trouve := true;
     for j := 1 to bo2nt do if sres[j] <> tnt [i,j] then trouve := false;
     if trouve then indice := i
     end
end;

begin
writeln ('* AVEZ VOUS UNE PHRASE A ANALYSER ?');
readln (reponse);
while (reponse = 'o') or (reponse = 'O') do
     begin
     writeln ('* VOUS AVEZ DECIDE D''ANALYSER UNE PHRASE.');
     writeln ('* QUEL EST LE PREMIER SOUS-RESEAU A PARCOURIR ?');
     for i := 1 to bo2nt do sres[i] := ' ';
     readln (sres);
     res := indice (sres);
     if res = 0
     then writeln ('* CE N''EST PAS UN NON-TERMINAL.')
     else begin
          new (pm);
          premot := pm;
          dermot := pm;
          pm^.suivant := nil;
          pm^.precedent := nil;
          for i := 1 to longm do pm^.mot[i] := ' ';
          pm^.score := 0;
          pm^.position := 0;
          new (p);
          premier := p;
          dernier := p;
          premot^.pelt := p;
          p^.suivant := nil;
          p^.precedent := nil;
          p^.premot := premot;
          p^.ps := 0;
          p^.s := 0;
          p^.e := 0;
          p^.pe := 0;
          parcours (res,resultat,0);
```

```
          if resultat
          then begin
               writeln ('* LA PHRASE A ETE RECONNUE.');
               writeln ('* VOICI L''ARBRE SYNTXIQUE :');
               writeln;
               arbre(adc);
               writeln
               end
          else writeln ('* LA PHRASE N''A PAS ETE RECONNUE.')
          end;
     writeln ('* AVEZ VOUS ENCORE UNE PHRASE A ANALYSER ?');
     pm := dermot;
     p := dernier;
     while pm <> nil do
          begin
          dermot := dermot^.precedent;
          dispose (pm);
          pm := dermot
          end;
     while p <> nil do
          begin
          dernier := dernier^.precedent;
          dispose (p);
          p := dernier
          end;
     readln (reponse)
     end
end;

(*************************************************************************)
(*                                                                       *)
(*                          MODULES DE NIVEAU 3                          *)
(*                          -------------------                          *)
(*                                                                       *)
(*************************************************************************)
(*************************************************************************)
(*                                                                       *)
(*                          MODULE PARCOURS                              *)
(*                                                                       *)
(*************************************************************************)

procedure parcours;
var position : integer;
    e        : integer;
    pe       : integer;
    finparc  : boolean;
begin
position := deb[res];
e := 0;
pe := -reseau[position];
finparc := false;
modnds (finparc,resultat,res,position,e,pe,resapp)
end;

(*************************************************************************)
(*                                                                       *)
(*                          MODULE REPRISE                               *)
(*                                                                       *)
(*************************************************************************)

procedure reprise;
var position : integer;
    e        : integer;
    pe       : integer;
    entree   : integer;
    arc      : integer;
```

```
    finparc  : boolean;
    finbr    : boolean;
begin
position := fin[res];
p := premier;
premier := premier^.suivant;
premier^.precedent := nil;
dispose (p);
entree := premier^.e;
position := reseau[position-entree];
resultat := false;
if (reseau[position] > 0) and (reseau[position] <= 1000)
then reprise (reseau[position],resultat,res)
else repbr (position,resultat);
if not resultat
then begin
     gestrep (res,position,resultat);
     if (resultat) and (reseau[position] >= 0)
     then begin
          arbre (position);
          finbr := false;
          finparc := false;
          modbr (finbr,finparc,resultat,position,e,pe,res);
          modnds (finparc,resultat,res,position,e,pe,resapp)
          end
     else
     if (resultat) and (reseau[position] < 0)
     then begin
          position := -reseau[position];
          e := rechent (position);
          position := position + e;
          pe := -reseau[position];
          finparc := false;
          modnds (finparc,resultat,res,position,e,pe,resapp)
          end
     end
else begin
     position := position + 1;
     position := -reseau[position];
     e := rechent(position);
     position := position + e;
     pe := -reseau[position];
     ndsproc(pe,e,resapp)
     end
end;

(*****************************************************************************)
(*                                                                           *)
(*                              SOUS-MODULES                                 *)
(*                                                                           *)
(* Ces modules ont ete separes des modules parcours-sous-reseau et reprise   *)
(* sous-reseau parceque les memes actions sont effectuees plusieurs fois a   *)
(* des endroits differents. En regroupant ainsi ces actions dans les modu-   *)
(* les modnds et modbr, on peut reduire la taille du code.                   *)
(*                                                                           *)
(*****************************************************************************)

procedure modnds;
var sortie : integer;
    finbr  : boolean;
begin
while not finparc do
     begin
     ndsproc (pe,e,resapp);
     if position = fin[res]
     then begin
```

```
          finparc := true;
          resultat := true
          end
     else begin
          sortie := premier^.s;
          position := reseau[position+sortie];
          finbr := false;
          modbr (finbr,finparc,resultat,position,e,pe,res)
          end
     end
end;

procedure modbr;
var arc : integer;
begin
while not finbr do
     begin
     arc := reseau[position];
     brlin (arc,resultat,res,position);
     if resultat
     then begin
          position := position + 1;
          arbre (position);
          if reseau[position] < 0
          then begin
               finbr := true;
               position := -reseau[position];
               e := rechent (position);
               position := position + e;
               pe := -reseau[position]
               end
          end
     else begin
          gestrep (res,position,resultat);
          if resultat
          then begin
               arbre (position);
               if reseau[position] < 0
               then begin
                    finbr := true;
                    position := -reseau[position];
                    e := rechent (position);
                    position := position + e;
                    pe := -reseau[position]
                    end
               end
          else begin
               finbr := true;
               finparc := true
               end
          end
     end
end;

(*****************************************************************************)
(*                                                                           *)
(*                          MODULES DE NIVEAU 4                              *)
(*                          -------------------                              *)
(*                                                                           *)
(*****************************************************************************)
(*****************************************************************************)
(*                                                                           *)
(*                          MODULE AFFICHAGE-ARBRE                           *)
(*                                                                           *)
(*****************************************************************************)
```

```
procedure arbre;
var arbinc : boolean;

procedure affich (res : integer ; position : integer ; var arbinc : boolean);
var pos    : integer;
    sortie : integer;
    finbr  : boolean;
    finres : boolean;
    arc    : elnt;
begin
finres := false;
p := p^.precedent;
pos := deb[res];
write ('( ');
while not (finres or arbinc) do
     begin
     if pos = fin[res]
     then begin
          finres := true
          end
     else begin
          while (p^.pe = 0) and (p <> premier) do
               begin
               p := p^.precedent
               end;
          sortie := p^.s;
          pos := reseau[pos + sortie];
          if (pos = position) and (p = premier)
          then arbinc := true
          else finbr := false;
          while not (finbr or arbinc) do
               begin
               if reseau[pos] = 0
               then write ('^ ')
               else
               if reseau[pos] <= 1000
               then affich (reseau[pos],position,arbinc)
               else
               if reseau[pos] <= 2000
               then begin
                    arc := ttl[reseau[pos] - 1000];
                    write (arc,' ')
                    end
               else
               if reseau[pos] <= 3000
               then begin
                    arc := ttv[reseau[pos] - 2000];
                    write (arc,' ')
                    end
               else begin
                    arc := ttg[reseau[pos] - 3000];
                    write (arc,' ')
                    end;
               pos := pos + 1;
               if (pos >= (position)) and (p = premier)
               then begin
                    arbinc := true
                    end;
               if reseau[pos] < 0
               then begin
                    finbr := true;
                    pos := -reseau[pos];
                    pos := pos + rechent (pos)
                    end
               end
          end;
```

```
     if not (finres or arbinc)
     then p := p^.precedent
     end;
if not arbinc
then begin
     arc := tnt[res];
     write (') * ',arc,' ')
     end
end;

begin
arbinc := false;
p := dernier;
affich (res,position,arbinc);
writeln
end;

(*****************************************************************************)
(*                                                                           *)
(*                          MODULE GESTION-REPRISES                          *)
(*                                                                           *)
(*****************************************************************************)

procedure gestrep;
var finrep : boolean;
    noeud  : boolean;
    ndent  : boolean;
    possib : boolean;
    sortie : integer;
    entree : integer;
begin
position := position - 1;
finrep := false;
while not finrep do
     begin
     noeud := false;
     while not noeud do
          begin
          if reseau[position] < 0
          then begin
               noeud := true
               end
          else
          if (0 < reseau[position]) and (reseau[position] <= 1000)
          then begin
               reprise (reseau[position],resultat,res);
               if resultat
               then begin
                    position := position + 1;
                    noeud := true;
                    finrep := true
                    end
               else begin
                    position := position - 1
                    end
               end
          else begin
               repbr (position,resultat);
               if resultat
               then begin
                    position := position + 1;
                    noeud := true;
                    finrep := true
                    end
               else begin
                    position := position - 1
```

```
                    end
               end
     end;
     if not finrep
     then begin
          position := -reseau[position];
          position := rechnd (position);
          if premier^.ps = -reseau[deb[res]]
          then ndent := true
          else ndent := false;
          if (premier^.suivant^.e = 0) and
             (premier^.suivant^.pe = 0)
          then possib := true
          else possib := false;
          if (ndent) and (not possib)
          then begin
               p := premier;
               premier := premier^.suivant;
               premier^.precedent := nil;
               dispose (p);
               finrep := true;
               resultat := false
               end
          else
          if possib
          then begin
               p := premier;
               premier := premier^.suivant;
               premier^.precedent := nil;
               dispose (p);
               sortie := premier^.s;
               position := reseau[position + sortie];
               finrep := true;
               resultat := true
               end
          else begin
               p := premier;
               premier := premier^.suivant;
               premier^.precedent := nil;
               dispose (p);
               entree := premier^.e;
               position := reseau[position - entree]
               end
          end
     end
end;

(*****************************************************************************)
(*                                                                           *)
(*                          MODULE BRANCHES-LINEAIRES                        *)
(*                                                                           *)
(*****************************************************************************)

procedure brlin;
type elt = array[1..bo2nt] of char;
var  nom : elt;

procedure branche (nom : elt ; var resultat : boolean ; position : integer);
var ch      : char;
    motvide : boolean;
    i       : integer;
    score   : integer;
    mot     : string[longm];
begin
writeln ('EST-CE QUE ',nom,' SUIT DANS LA PHRASE ?');
readln (ch);
```

```
resultat := false;
if (ch = 'O') or (ch = 'o')
then begin
     for i := 1 to longm do mot[i] := ' ';
     writeln ('VOUS DEVEZ INTRODUIRE LES ',nom,' RECONNUS ET LES SCORES.');
     writeln ('MOT :');
     readln (mot);
     motvide := true;
     for i := 1 to longm do if mot[i] <> ' ' then motvide := false;
     while not motvide do
          begin
          resultat := true;
          writeln ('SCORE :');
          readln (score);
          if (premier <> premot^.pelt) or
             (position <> premot^.position) or
             ((premier = premot^.pelt) and
              (position = premot^.position) and
              (score >= premot^.score))
          then begin
               new (pm);
               premier^.premot := pm;
               pm^.suivant := premot;
               premot^.precedent := pm;
               premot := pm;
               premot^.precedent := nil;
               premot^.mot := mot;
               premot^.score := score;
               premot^.position := position;
               premot^.pelt := premier
               end
          else begin
               pmint := premot;
               while (position = pmint^.position) and
                     (premier = pmint^.pelt) and
                     (score < pmint^.score) do
                    begin
                    pmint := pmint^.suivant
                    end;
               new (pm);
               pm^.suivant := pmint;
               pm^.precedent := pmint^.precedent;
               pmint^.precedent^.suivant := pm;
               pmint^.precedent := pm;
               pm^.mot := mot;
               pm^.score := score;
               pm^.position := position;
               pm^.pelt := premier
               end;
          writeln ('MOT :');
          for i := 1 to longm do mot[i] := ' ';
          readln (mot);
          motvide := true;
          for i := 1 to longm do if mot[i] <> ' ' then motvide := false
          end
     end
end;

begin
if arc < 0 then resultat := false
else
if arc = 0 then resultat := true
else
if arc <= 1000 then parcours (arc,resultat,res)
else
if arc <= 2000
```

```
then begin
     arc := arc - 1000;
     nom := ttl[arc];
     branche (nom,resultat,position)
     end
else
if arc <= 3000
then begin
     arc := arc - 2000;
     nom := ttv[arc];
     branche (nom,resultat,position)
     end
else begin
     arc := arc - 3000;
     nom := ttg[arc];
     branche (nom,resultat,position)
     end
end;

(*****************************************************************************)
(*                                                                           *)
(*                          MODULE REPRISE-BRANCHES                          *)
(*                                                                           *)
(*****************************************************************************)

procedure repbr;
begin
if (premier = premot^.suivant^.pelt) and
   (position = premot^.suivant^.position)
then begin
     resultat := true;
     pm := premot;
     premot := premot^.suivant;
     premot^.precedent := nil;
     premier^.premot := premot;
     dispose (pm)
     end
else begin
     resultat := false;
     if (premier = premot^.suivant^.pelt)
     then begin
          pm := premot;
          premot := premot^.suivant;
          premot^.precedent := nil;
          premier^.premot := premot;
          dispose (pm)
          end
     else begin
          pm := premot;
          premot := premot^.suivant;
          premot^.precedent := nil;
          premier^.premot := nil;
          dispose (pm)
          end
     end
end;

(*****************************************************************************)
(*                                                                           *)
(*                          MODULE NOEUDS-PROCEDURAUX                        *)
(*                                                                           *)
(*   Ce module a ete defini comme externe pour augmenter l'independance de   *)
(*   l'analyseur syntaxique vis vis du reseau particulier a parcourir.       *)
(*                                                                           *)
(*****************************************************************************)
```

```
(*****************************************************************************)
(*                                                                           *)
(*                          PROGRAMME PRINCIPAL                              *)
(*                          -------------------                              *)
(*                                                                           *)
(*****************************************************************************)

begin
analyse
end.
```

```
(*****************************************************************************)
(*                                                                           *)
(*                            ANALYSEUR-SYNTAXIQUE                           *)
(*                            ====================                           *)
(*                                                                           *)
(*   Cette partie de l'analyseur ne comprend que la description des proce-   *)
(*   dures associes aux differents noeuds proceduraux. Elle a ete separee    *)
(*   du code de l'analyseur proprement dit pour rendre ce dernier plus in-   *)
(*   dependant vis a vis du reseau particulier a parcourir.                  *)
(*                                                                           *)
(*****************************************************************************)
(*                                                                           *)
(* AUTEUR                        : MOUSEL Pierre                             *)
(* DATE DERNIERE MODIFICATION    : 13/12/82                                  *)
(*                                                                           *)
(*****************************************************************************)

subprogram noeuds(input,output);
(*$h=40000*)

(*****************************************************************************)
(*                                                                           *)
(*                  DECLARATION DES DONNEES GLOBALES                         *)
(*                  --------------------------------                         *)
(*                                                                           *)
(*****************************************************************************)

const bo1nt = 20;
      bo2nt = 4;
      bo1tv = 20;
      bo2tv = 4;
      bo1tl = 30;
      bo2tl = 4;
      bo1tg = 20;
      bo2tg = 4;
      bores = 2000;
      longm = 20;
type  elnt     = array [1..bo2nt] of char;
      eltv     = array [1..bo2tv] of char;
      eltl     = array [1..bo2tl] of char;
      eltg     = array [1..bo2tg] of char;
      chaine   = string [bo2nt];
      pointelt = ^elt;
      pointmot = ^mots;
      elt      = record
                   suivant   : pointelt;
                   ps        : integer;
                   s         : integer;
                   e         : integer;
                   pe        : integer;
                   premot    : pointmot;
                   precedent : pointelt
                   end;
      mots     = record
                   suivant   : pointmot;
                   mot       : string[longm];
                   score     : integer;
                   position  : integer;
                   pelt      : pointelt;
                   precedent : pointmot
                   end;
var   adc     : integer;
      nbnt    : integer;
      nbtv    : integer;
      nbtl    : integer;
```

```
      nbtg     : integer;
      res      : integer;
      sres     : chaine;
      tnt      : array [1..bolnt] of elnt;
      ttv      : array [1..boltv] of eltv;
      ttl      : array [1..boltl] of eltl;
      ttg      : array [1..boltg] of eltg;
      deb      : array [1..bolnt] of integer;
      fin      : array [1..bolnt] of integer;
      reseau   : array [1..bores] of integer;
      p        : pointelt;
      premier  : pointelt;
      dernier  : pointelt;
      pm       : pointmot;
      pmint    : pointmot;
      dermot   : pointmot;
      premot   : pointmot;

(*************************************************************************)
(*                                                                       *)
(*                          LISTE DES RPOCEDURES                         *)
(*                          --------------------                         *)
(*                                                                       *)
(*************************************************************************)

procedure analyse ; forward;
procedure lecres  ; forward;
procedure anaphr  ; forward;
procedure parcours (    res      : integer;
                    var resultat : boolean;
                        resapp   : integer); forward;
procedure reprise  (    res      : integer;
                    var resultat : boolean;
                        resapp   : integer); forward;
procedure modnds   (    finparc  : boolean;
                    var resultat : boolean;
                        res      : integer;
                        position : integer;
                        e,pe     : integer;
                        resapp   : integer); forward;
procedure modbr    (var finbr    : boolean;
                    var finparc  : boolean;
                    var resultat : boolean;
                    var position : integer;
                    var e,pe     : integer;
                        res      : integer); forward;
procedure ndsproc  (pe : integer ; e : integer ; resapp : integer); forward;
procedure brlin    (    arc      : integer;
                    var resultat : boolean;
                        res      : integer;
                        position : integer); forward;
procedure repbr    (    position : integer;
                    var resultat : boolean); forward;
procedure gestrep  (    res      : integer ;
                    var position : integer ;
                    var resultat : boolean); forward;
procedure arbre    (position : integer); forward;

(*************************************************************************)
(*                                                                       *)
(*                        MODULE NOEUDS-PROCEDURAUX                      *)
(*                                                                       *)
(*************************************************************************)

procedure ndsproc;
```

```
procedure creel (proc : integer ; ns : integer);
begin
new (p);
p^.suivant := premier;
premier^.precedent := p;
premier := p;
premier^.ps := proc;
premier^.s := ns;
premier^.e := 0;
premier^.pe := 0;
premier^.premot := nil;
premier^.precedent := nil;
end;

procedure noeud (proc : integer ; nbs : integer);
var i : integer;
begin
if nbs = 0
then creel (proc,nbs)
else for i := nbs downto 1 do creel (proc,i)
end;

begin
premier^.pe := pe;
premier^.e := e;
case pe of
     1 : noeud (1,3);
     2 : noeud (2,2);
     3 : noeud (3,2);
     4 : noeud (4,2);
     5 : noeud (5,4);
     6 : noeud (6,4);
     7 : noeud (7,1);
     8 : noeud (8,0);
     9 : noeud (9,3);
     10 : noeud (10,1);
     11 : noeud (11,3);
     12 : noeud (12,1);
     13 : noeud (13,3);
     14 : noeud (14,5);
     15 : noeud (15,3);
     16 : noeud (16,1);
     17 : noeud (17,0);
     18 : noeud (18,2);
     19 : noeud (19,0)
     end
end.
```